Universidad Tecnológica de Pereira

Facultad de Ciencias Empresariales

Harold Stiven Cardona Zapata

Aplicación del algoritmo genético con el fin de optimizar las tareas operativas de los ambientes de trabajo tipo job-shop

Application of the genetic algorithm in order to optimize the operational tasks of job-shop type work environments

Ingenería industrial

Directora: Eliana Mirledy Toro Ocampo, Ph.D.

Programa: Ingeniería industrial

Pereira, 2019

Universidad Tecnológica de Pereira

Facultad de Ciencias Empresariales

Harold Stiven Cardona Zapata

Aplicación del algoritmo genético con el fin de optimizar las tareas operativas de los ambientes de trabajo tipo job shop

Application of the genetic algorithm in order to optimize the operational tasks of job shop type work environments

Ingenería industrial

Directora: Eliana Mirledy Toro Ocampo, Ph.D.

Programa: Ingeniería industrial

Pereira, 2019

A mis padres

Agradecimientos

A todos aquellos que me han ayudado en este proceso.

Índice

Índice de figuras

Índice de tablas

1. Resumen

Una gran cantidad de investigación se ha centrado en resolver el problema de secuenciamiento de tareas en ambientes Job Shop o JSP (por sus siglas en inglés) en los últimos años, hecho que ha dado como resultado una gran variedad de enfoques que giran alrededor de los distintos métodos de solución empleados para este tipo de problema. Con el fin de adaptar de forma ágil el JSP a alguna técnica de solución para resolverlo en un tiempo computacional adecuado y de forma óptima, se propone el desarrollo e implementación de la técnica metaheurística conocida como el algoritmo genético en el lenguaje de programación Python integrando Microsoft Excel, considerando como medida de efectividad, la optimización del tiempo total requerido para terminar todas las tareas o makespan. Con el fin de validar la metodología propuesta e implementada se evalúan 5 casos de prueba de la literatura especializada a partir de los cuales se puede medir tanto la eficiencia computacional como la precisión del algoritmo para obtener los resultados.

2. Abstract

A large number of research has been focused on solving the task sequencing problem in Job Shop environments (JSP) in the last years, which has resulted in a wide variety of approaches that revolve around different solution methods used for this type of problem. In order to adapt the JSP in an agile way to some solution technique to solve it in an adequate and optimal computational time, it's proposed the development and implementation of the metaheuristic technique known as the genetic algorithm in the Python programming language integrating Microsoft Excel, considered as a measure of precision, the optimization of the total time required to complete all tasks or makepan. In order to validate the proposed and implemented methodology, 5 test cases of the specialized literature are evaluated from which both the computational efficiency and the accuracy of the algorithm can be measured to obtain the results.

3. Introducción

Actualmente las empresas deben ser muy competitivas para poder perdurar en el mercado, por tal razón estas deben ser flexibles, tener una buena aceptación al cambio y aplicar nuevas herramientas con el propósito de volver sus procesos productivos más eficientes, logrando de esta manera satisfacer todas las necesidades del mercado que cada día es más exigente. Esta situación conlleva a que las empresas, especialmente las de producción, indaguen sobre la manera óptima de programar las tareas de tal forma que se reduzcan los tiempos de fabricación y se logre una mayor utilización de los recursos. Una forma de brindar solución a esta problemática es a partir de la implementación de una nueva metodología que permita optimizar el mínimo tiempo total requerido para el procesamiento de todas sus labores.

Entre las herramientas que responden a estos requisitos se encuentra el secuenciamiento de tareas, cuyo objetivo se centra en encontrar el orden óptimo de elaboración de los diferentes procesos requeridos por N trabajos distintos en M máquinas de acuerdo con ciertas medidas de desempeño global. Existen diferentes objetivos que se tienen en cuenta al planear una secuenciación; el más frecuentado es el de incrementar la utilización de los recursos, puesto que la utilización de estos es inversamente proporcional al tiempo requerido para terminar todas las tareas (makespan); otro objetivo consiste en reducir el inventario de productos en proceso, es decir, minimizar el número promedio de tareas en espera a realizarse mientras las máquinas están ocupadas.

Estos problemas de planeación de la producción se resuelven empleando técnicas heurísticas o metaheurísticas debido a que la naturaleza misma del problema genera una explosión combinatorial en el espacio de soluciones [1]. El proceso de planeamiento está compuesto por varios componentes funcionales los cuales contienen múltiples planes de soporte y el modelo típico de planificación se basa en pronósticos, plan maestro de producción, plan de materiales, plan de producción y programa de operaciones, como se especifica en la figura

1. Teniendo en cuenta lo anterior, se puede notar que la planeación y la programación de la producción son de vital importancia en las empresas, pues se refieren a las actividades del día a día que tienen que ser resueltas de forma inmediata.

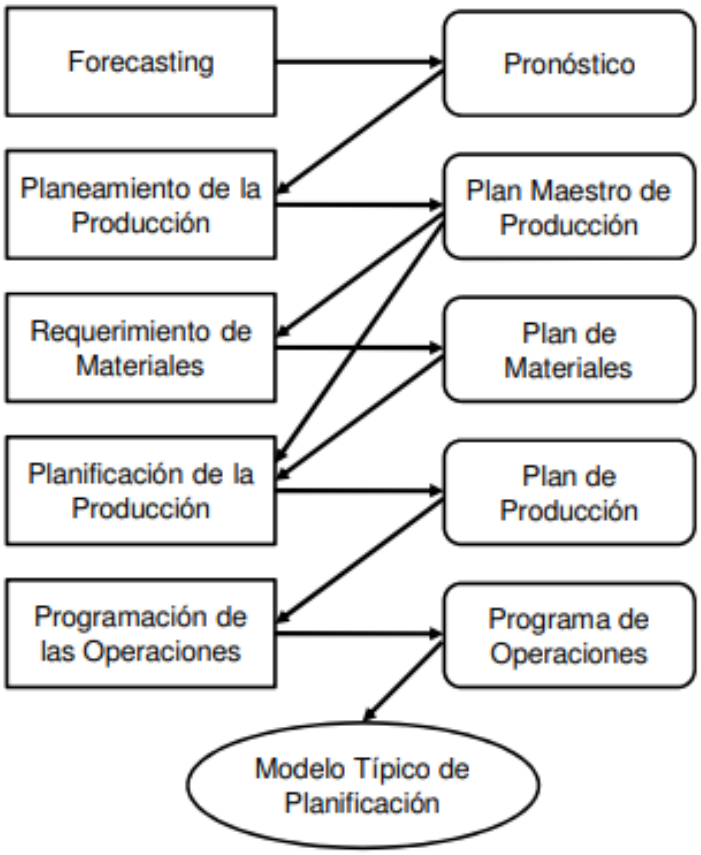

Figura 1: Modelo Típico de Planificación [1]

La investigación en la teoría de la programación ha evolucionado a lo largo de los últimos años y ha sido objeto de mucha literatura significativa con técnicas que van desde reglas de despacho sin refinar hasta algoritmos altamente sofisticados de ramificación, enlaces paralelos y heurísticas basadas en cuellos de botella. No es sorprendente que los enfoques se hayan formulado a partir de un espectro diverso de investigadores que van desde científicos de gestión a trabajadores de producción. Sin embargo, con el advenimiento de nuevas metodologías, como las redes neuronales y la computación evolutiva, los

investigadores de campos como la biología, la genética y la neurofisiología también se han convertido en colaboradores habituales de la teoría de la programación, enfatizando en la naturaleza multidisciplinaria de este campo [3].

Uno de los modelos más populares en la teoría de la programación es el Job-Shop Problem (JSP), ya que se considera ser una buena representación del dominio general y se ha ganado una reputación por ser notoriamente difícil de resolver, probablemente sea el modelo más estudiado y bien desarrollado en la teoría de la programación determinista, que sirve como un banco de pruebas comparativo para diferentes técnicas de solución antiguas y nuevas, y como también está fuertemente motivado por los requisitos prácticos, vale la pena la compresión de este modelo a fondo.

Este problema, al igual que muchos otros en el campo de secuenciación de tareas se considera de difícil solución. Esta característica hace que el problema sea permutacional, es decir, cada una de las combinatorias que pueden formarse deben contener las N tareas a procesar y una permutación se diferencia de otra únicamente en el orden de ubicación de los elementos, obteniéndose de esta manera que el número de permutaciones posibles es $N!$. En la literatura especializada, los problemas combinatoriales están catalogados como $NP - Hard$, dado que no se cuentan con algoritmos de orden polinomial a partir de los cuales se pueda resolver el problema de forma óptima y en el menor tiempo posible. Se dice que un problema es $NP - Hard$ cuando se demuestra que cualquier algoritmo de solución tiene un tiempo de ejecución que aumenta, en el peor de los casos, exponencialmente con el tamaño del problema [4].

Durante los últimos años, la investigación sobre la programación de tareas, en especial su aplicación de forma industrial, ha aumentado en importancia a causa de las demandas de la industria. Si bien se ha avanzado mucho en el frente académico, continúan las dudas sobre la transferencia de tecnología para cumplir con los requisitos de flexibilidad de las instalaciones de producción modernas. La programación tipo Job Shop ha tenido mucho

atractivo en la literatura, ya que la solución se hace más difícil a través del requisito de satisfacer las demandas tanto de la producción por lotes como de la producción continua. En investigaciones anteriores, se ha trabajado en problemas de programación de máquinas con la esperanza de encontrar una solución óptima o casi óptima para problemas de gran complejidad.

Una función importante de la programación JSP es la coordinación y el control de actividades complejas, tanto la asignación óptima de recursos como la secuencia en el desempeño de esas actividades. La variedad de reglas de programación y procedimientos para ciertos tipos de talleres han evolucionado a partir de estos esfuerzos. Las técnicas de planificación y control de redes han encontrado una amplia aplicación a los problemas de programación asociados con las actividades del proyecto. También se han propuesto numerosos procedimientos para asignar estaciones de trabajo óptimas o casi óptimas a las líneas de montaje. Últimamente la mayoría de los estudios han recurrido a nuevas técnicas de resolución como la simulación y las técnicas heurísticas y metaheurísticas. Estos métodos han demostrado ser un importante paso en la solución del modelo JSP con menos esfuerzo computacional y resultados más poderosos.

El propósito de este documento es revisar los métodos y técnicas que se han utilizado en la programación de los problemas JSP y proponer una solución a partir de la implementación de una metaheurística conocida como el algoritmo genético para resolver los casos propuestos de manera eficiente haciendo uso de la integración del lenguaje de programación Python y Microsoft Excel con el objetivo de proporcionar una herramienta eficiente y de fácil acceso a las pequeñas y medianas empresas de la región.

4. Objetivo de la programación Job Shop

La programación de tareas es el área en los sistemas de control de producción en la que las actividades planificadas previamente, como los cronogramas de producción y los niveles de inventario, se proyectan en una escala de tiempo detallada. La asignación detallada de trabajos y materiales a recursos humanos y físicos, personal y máquinas, tiene lugar en la programación. Los horarios se basan en la planificación agregada o en los horarios maestros, los tamaños de lote óptimos establecidos y el conocimiento de los recursos disponibles. En este proceso se intenta encontrar programas detallados que sean óptimos con respecto a las fechas de vencimiento de las reuniones, la alta utilización de la máquina, el bajo costo unitario y otros fines posibles. Los resultados de las actividades de programación se retroalimentan a las otras áreas de planificación y control para mejorar su toma de decisiones [5].

Para diferenciar entre horarios y seleccionar el mejor, se deben tener algunas medidas de efectividad, como en otras áreas donde se quiere optimizar, con las que se puedan comparar las diferentes soluciones. En general, se desea minimizar la duración del tiempo de operación, como el tiempo total de procesamiento, el tiempo de finalización de ciertos productos, el tiempo promedio de finalización, el tiempo total del proyecto, minimizar el tiempo de inactividad, o minimizar ciertos costos, como el costo unitario de producción o el costo total. La idea subyacente para todos estos objetivos es la maximización de la ganancia.

Examinando más detenidamente las diferentes medidas de efectividad, se encuentra que algunos de ellos no son aplicables para ciertos problemas y que, lo que es peor, muchos de ellos son contradictorios. Aquí surge el dilema de la programación, que es particularmente evidente en la producción tipo Job Shop. Los siguientes puntos son ejemplos de la contradicción en la operación de programación:

- Disminuir el tiempo promedio en proceso de las órdenes de trabajo, disminuyendo

así el inventario en proceso y aumentando la probabilidad de cumplir las fechas de vencimiento.

- Aumentar el grado de utilización de equipos, aumentando así el retorno de la inversión física

El logro del primer objetivo llevaría a la selección del programa con el tiempo de procesamiento más pequeño para todos los productos, como el tiempo de procesamiento, el tiempo de transporte, el tiempo de espera y el tiempo de configuración. Este objetivo se enfoca en los trabajos a realizar e implica moverlos rápidamente a través del proceso de producción. Para lograr el segundo objetivo, se debe elegir el programa que maximice la utilización de la capacidad existente. Este objetivo se centra en las máquinas e implica la disposición de los trabajos para adaptarse a las maquinas.

Se ve fácilmente que un programa que es óptimo con respecto al tiempo total en proceso no tiene que ser óptimo con respecto a la utilización de la capacidad existente. Sin embargo, se debe tener en cuenta que la contradicción de estos objetivos existe solo para horizontes de planificación bastante cortos o si la información sobre órdenes futuras es muy incierta. A largo plazo, el objetivo de costo mínimo, incluido el costo de capital y el costo de inventario, incluye la mayoría de los otros subobjetivos del inventario mínimo en proceso o la utilización máxima de la capacidad de la máquina. De los dos objetivos principales mencionados anteriormente, se pueden derivar varias medidas secundarias de efectividad que tienen en cuenta algunos aspectos del problema general o se centran en factores importantes que influyen en el resultado total:

1. Minimizar el tiempo que las instalaciones están ocupadas.
2. Minimizar el tiempo total de inactividad.
3. Minimizar el tiempo total de espera de los productos.
4. Minimice el retraso total, es decir, el tiempo que se tarda en terminar los productos después de la fecha de entrega.

5. Descripción de la programación de Job Shop

Los ambientes de tipo Job Shop implican la fabricación de cantidades discretas, esto conlleva a la producción donde las unidades son procesadas como entidades individuales o en lotes pequeños y la programación generalmente se controla mediante una hoja de enrutamiento o un proceso de pedido corto en lugar de un sistema de línea de ensamblaje. El equipo de producción en el taller generalmente tiene una naturaleza de propósito general para proporcionar la flexibilidad necesaria por la variación en tamaño, forma, cantidad, precisión y tipo de producto. Por lo general, las máquinas similares se agrupan en centros de trabajo, y originalmente cada máquina puede realizar una variedad de tareas. El problema de la programación de trabajos consiste en determinar el orden o la secuencia en que las máquinas procesarán los trabajos para optimizar el rendimiento.

5.1. Características del JSP

La naturaleza de una amplia variedad de productos y las plantas en las que se producen proporcionan ciertas características comunes de los ambientes tipo Job Shop las cuales se describen a continuación:

1. En cualquier momento hay una gran cantidad de pedidos en varias etapas de finalización.

2. Las órdenes hacen demandas conflictivas sobre las instalaciones y la mano de obra disponible.

3. Cada orden difiere en cierta medida. Por lo tanto, es difícil predecir el tiempo requerido para completar las operaciones.

4. El flujo de trabajo es intermitente y las órdenes se pueden desviar.

5. Por lo general, hay una cola de trabajo en cada máquina y, a menudo, es difícil determinar qué orden debe tener la cola.

6. Hay muchos cambios resultantes de desecho, retrabajo, avería de la máquina, escasez de materiales, cambios de ingeniería.

7. Se realiza un esfuerzo considerable para determinar el estado de los pedidos y para acelerar los pedidos a través de varios departamentos.

8. Los horarios y las cargas de la tienda rara vez se alteran debido a la gran carga de trabajo administrativo necesaria para realizar las modificaciones. En resumen, el taller es complejo e impredecible. El control cercano rara vez se establece.

Al desarrollar un sistema de programación Job Shop, es importante lograr un equilibrio adecuado entre el control y la flexibilidad para mejorar los problemas en lugar de eliminarlos por completo. Existen cuatro factores sirven para describir y clasificar un problema específico de la programación de este tipo:

1. El patrón de llegada del trabajo. Si los trabajos llegan simultáneamente al taller que está inactivo e inmediatamente disponible para trabajar, se dice que el problema de la programación es estático. Si los trabajos llegan de forma intermitente, posiblemente de acuerdo con un proceso estocástico, el problema de programación es dinámico.

2. Es necesario especificar el número de máquinas que componen el taller.

3. Se debe especificar el proceso de flujo de trabajos a través de la máquina. Si todos los trabajos siguen el mismo enrutamiento, entonces la tienda es una taller tipo flow shop. El extremo opuesto es el taller tipo Job Shop distribuido aleatoriamente, en el que los trabajos no siguen una secuencia común de operaciones.

4. El criterio para evaluar el desempeño del taller de trabajo juega un papel crítico en el proceso de programación.

5.2. Resultados de los problemas de programación

El resultado de la programación de un taller de trabajo tipo Job Shop puede indicarse de la siguiente manera:

1. Determinar la postura estratégica a largo plazo del taller en relación con el mercado, en particular, para acordar la combinación de productos objetivo y la configuración correspondiente de las capacidades del taller.

2. Planificar y controlar detalladamente el tiempo de producción en el taller para lograr una producción eficiente, en particular, para reducir los tiempos de entrega y los bajos costos de configuración e inventario en proceso.

3. Negociar el tiempo de las entregas con los clientes de manera realista que refleje la presencia de otros pedidos, las capacidades del taller y el costo de lograr un tiempo determinado, así como el valor del tiempo para el cliente.

4. Planificar la configuración del taller, en particular la asignación de la mano de obra a fin de realizar el trabajo requerido de manera eficiente.

5. Negociar las entregas de los proveedores sobre la base de un plan de producción consistente, teniendo en cuenta todos los pedidos que se producirán y las capacidades del taller.

6. Programar y controlar otras actividades de preproducción, como el trabajo de ingeniería; coordinar estas actividades para llevar a cabo la producción según lo previsto.

7. Realizar estas operaciones de planificación y planificación sobre la base de la información que llega al formulario de forma irregular a través del tiempo, lo que permite la incertidumbre sobre el futuro y la aparición de alteraciones no controladas e impredecibles en la tienda, por parte de proveedores y clientes y de hecho en el propio sistema de programación.

6. Fundamentación teórica

6.1. Descripción del problema

Se considera una planta donde los trabajos son procesados por máquinas. Cada trabajo consiste en un cierto número de operaciones. Cada operación debe ser realizada por una máquina dedicada y requiere un tiempo de procesamiento predefinido. La secuencia de operación se prescribe para cada trabajo en una receta de producción, imponiendo restricciones estáticas en la programación. Por lo tanto, cada trabajo tiene su propio pedido de máquina y no existe ninguna relación entre los pedidos de máquina de cualquiera de los dos trabajos [6].

En el problema JSS, un conjunto J de n trabajos $J_1, J_2, J_3, ...J_n$ debe procesarse en un conjunto M de m máquinas diferentes $M_1, M_2, M_3, ...M_m$. El trabajo J_j consiste en una secuencia de m_j operaciones $O_{j1}, O_{j2}, O_{j3}, ..., O_{jmj}$, que deben programarse en este orden.

Además, cada operación puede ser procesada solo por una máquina entre las m disponibles. La operación O_{jk} tiene un tiempo de procesamiento P_{jk}. El objetivo es encontrar una secuencia operativa para cada máquina, como minimizar una función particular de los tiempos de finalización del trabajo, y de tal manera que dos operaciones nunca se procesen en la misma máquina en cualquier momento [7].

Las siguientes suposiciones se hacen a lo largo de este capítulo:

1. Hay N trabajos que requieren procesamiento en máquinas M (Trabajo i = 1,2,...,N; Máquina m = 1,2,.., M).
2. Cada trabajo requiere cada máquina y ninguna máquina procesa más de una vez.
3. Una máquina puede procesar solo un trabajo en un momento dado.
4. Sólo hay una máquina de cada tipo.

5. Todas las operaciones, una vez iniciadas, deben completarse sin interrupción.

6. Se supone que los tiempos de procesamiento se conocen sin error.

7. Los tiempos de procesamiento son independientes entre sí y también del orden en que se procesan.

8. El tiempo de configuración y el tiempo requerido para transportar trabajos entre máquinas es cero.

En el JSP estándar, el problema de la programación se describe a continuación. Se dan un conjunto de trabajos y un conjunto de máquinas. Cada máquina puede manejar como máximo un trabajo a la vez. Cada trabajo consiste en una cadena de operaciones, cada una de las cuales debe procesarse durante un período de tiempo ininterrumpido de una longitud determinada en una máquina determinada. Esta estandarización ayuda a abarcar una amplia gama de instancias de problemas. El propósito es encontrar una programación, es decir, una asignación de las operaciones a intervalos de tiempo en las máquinas, que tenga una longitud mínima. El problema permite una serie de formulaciones matemáticas relativamente sencillas. Además, es extremadamente difícil de resolver de manera óptima. Se presenta una encuesta exhaustiva sobre los diferentes problemas de programación y sus algoritmos y complejidades [8].

6.2. Cálculo del Makespan

Minimizar el makespan en un ambiente de trabajo Job Shop puede ser representado mediante un gráfico disyuntivo [2], este método es descrito a continuación:

Se considera un gráfico dirigido G con un conjunto de nodos N y dos conjuntos de arcos A y B. Los nodos N corresponden a todas las operaciones (i, j) que deben realizarse en los n trabajos. Los llamados arcos conjuntivos (sólidos) A representan las rutas de los trabajos. Si el arco $(i, j) \rightarrow (k, j)$ es parte de A, entonces el trabajo j debe procesarse en la máquina I antes de procesarse en la máquina k, es decir, la operación (i, j) precede a

la operación (k, j).

Dos operaciones que pertenecen a dos trabajos diferentes y que deben procesarse en la misma máquina están conectadas entre sí por dos llamados arcos disyuntivos (rotos) que van en direcciones opuestas. Los arcos disyuntivos B forman m cliques de arcos dobles, un clique para cada máquina. (un clique es una teoría de gráfico de terminación que se refiere a una gráfica en la cual dos nodos están conectados entre sí; en este caso, cada conexión dentro de un clique consiste en un par de arcos disyuntivos). Todas las operaciones (nodos) en el mismo clique deben ser procesadas en la misma máquina. Todos los arcos que emanan de un nodo, tanto conjuntivos como disyuntivos, tienen la longitud del tiempo de procesamiento de la operación que representa ese nodo. Además, hay una fuente U y un sumidero V, que son nodos ficticios. El nodo fuente U tiene n arcos conjuntivos que emanan a las primeras operaciones de los n trabajos y el nodo sumidero V tiene n arcos conjuntivos procedentes de todas las últimas operaciones. Los arcos que emanan de la fuente tienen una longitud cero (ver Figura 2). Este gráfico se denota por $G = (N, A, B)$.

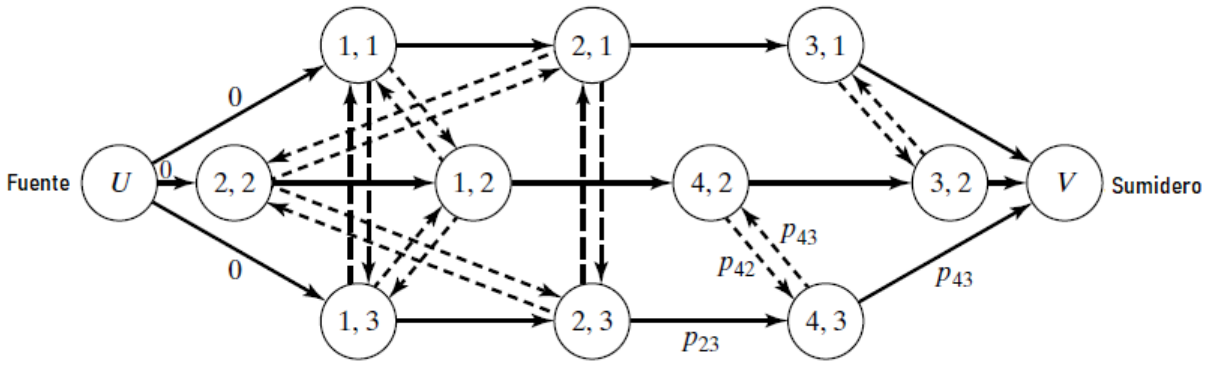

Figura 2: Gráfico dirigido para JSP con makespan como objetivo [2]

Un programa factible corresponde a una selección de un arco disyuntivo de cada par, de modo que el gráfico dirigido resultante sea acíclico. Esto implica que una selección de arcos disyuntivos de un clique tiene que ser acíclico. Dicha selección determina la secuencia en la que se realizarán las operaciones en esa máquina. Se puede argumentar que una selección de un clique debe ser acíclica de la siguiente manera: si hubiera un ciclo dentro de un clique, no habría sido posible una secuencia factible de las operaciones en la máquina correspondiente. Puede no ser inmediatamente obvio por qué no debería haber ningún ciclo formado por arcos conjuntivos y arcos disyuntivos de cliques diferentes. Sin embargo, dicho ciclo correspondería también a una situación que no es factible. Por ejemplo, (h, j) y (i, j) denotan dos operaciones consecutivas que pertenecen al trabajo j y (i, k) y (h, k) denotan dos operaciones consecutivas que pertenecen al trabajo k. Si bajo una programación determinada, la operación (i, j) precede a la operación (i, k) en la máquina i y la operación (h, k) precede a la operación (h, j) en la máquina h, entonces el gráfico contiene un ciclo con cuatro arcos, dos arcos conjuntivos y dos arcos disyuntivos de diferentes cliques. Tal horario es físicamente imposible. Resumiendo, si D denota el subconjunto de los arcos disyuntivos seleccionados y el gráfico $G(D)$ está definido por el conjunto de arcos conjuntivos y el subconjunto D, entonces D corresponde a un programa factible si y solo si $G(D)$ no contiene ciclos.

La duración de un horario factible está determinada por la ruta más larga en $G(D)$ desde la fuente U hasta el sumidero V. Esta ruta más larga consiste en un conjunto de operaciones de las cuales la primera comienza en el tiempo 0 y la última finaliza en el momento del makespan. Cada operación en esta ruta es seguida inmediatamente por la siguiente operación en la misma máquina o la siguiente operación del mismo trabajo en otra máquina. El problema de minimizar el makespan se reduce a encontrar una selección de arcos disyuntivos que minimice la longitud de la ruta más larga (es decir, la ruta crítica del problema).

Existen varias formulaciones de programación matemática para el problema Job Shop sin recirculación, incluidas varias formulaciones de programación de enteros. Sin embargo, la formulación más utilizada es la denominada formulación de programación disyuntiva. Esta formulación de programación disyuntiva está estrechamente relacionada con la representación gráfica disyuntiva del JSP.

7. Estado del arte

7.1. Procedimientos de optimización

En procedimientos exactos el requisito de tiempo para encontrar una solución aumenta exponencialmente o como un polinomio de alto grado para un aumento lineal en el tamaño del problema, excepto para las versiones restringidas (casos especiales) de JSP.

7.1.1. Métodos eficientes

Los métodos eficientes resuelven un problema tipo Job Shop de manera óptima con un tiempo requerido de procesamiento que aumenta polinomialmente con respecto al tamaño de su entrada. Estos métodos construyen una solución óptima a partir de los datos del problema siguiendo un conjunto simple de reglas que determinan exactamente el orden de procesamiento.

El primer ejemplo de un método eficiente y el primer trabajo en la teoría de la programación de tareas lo presenta Johnson [6], quien desarrolla un algoritmo eficiente para un problema tipo flow shop de dos máquinas simple que minimiza el makespan. Otros métodos eficiente desarrollados para el Job Shop incluyen el propuesto por Akers [9] y Hefetz y Adiri [10] donde presentan un enfoque en el que todas las operaciones son de tiempo de procesamiento de unidades discretas.

A pesar de que los casos especiales anteriores de Job Shop se han resuelto de manera óptima, existen $(n!)^M$ soluciones posibles. Así, un problema de $20x10$ (20 trabajos y 10 máquinas) tiene 726510183 posibles soluciones en principio, que exceden la edad postulada del universo en microsegundos. Aunque muchas de estas soluciones no serán factibles debido a la precedencia y las restricciones disyuntivas, la enumeración completa de todas las secuencias factibles para identificar la óptima no es práctica. Como resultado de esta intratabilidad, se considera que el problema de Job Shop pertenece a la clase de problemas

de decisión que son del tipo NP [11].

NP implica polinomio no determinista, lo que significa que no es posible resolver un caso arbitrario en tiempo polinomial a menos que $P = NP$ [12]. P es una subclase de NP y consiste en el conjunto de problemas que se pueden resolver de manera determinista en el tiempo polinomial (por ejemplo, los casos especiales de Job Shop antes mencionados pertenecen a la clase P). Un problema de decisión es NP-Completo si pertenece al conjunto NP y es tan difícil como cualquier otro problema en NP. La variante de optimización correspondiente de tal instancia se dice que es $NP - Hard$. Algunos problemas de $NP - Hard$ se pueden resolver polinomialmente con respecto a diferentes representaciones de la entrada, por ejemplo, el problema de tardanzas totales de una máquina tiene un tiempo de ejecución que es polinomial en la suma de los tiempos de procesamiento.

Es así que solo basta con una variación muy leve en la definición del problema para que estos mismos, a pesar de ser solucionados de manera eficiente, se conviertan rápidamente en $NP - Hard$ o fuertemente $NP - Hard$. La intratabilidad del problema de Job Shop se destaca aún más por el hecho de que un problema de $10x10$ (10 trabajos y 10 máquinas) propuesto por Fisher y Thompson [13] solo podría ser resuelto mediante el método propuesto por Carlier y Pinson [14] a pesar de que se han probado todos los algoritmos posibles.

A pesar del progreso logrado por estos trabajos recientes, no se pueden encontrar métodos eficientes para las instancias reales de Job Shop, incluso French [15] predijo que nunca se desarrollarían algoritmos eficientes para la mayoría de los problemas de programación. Como resultado, el enfoque de la investigación de optimización se ha convertido en enfoques enumerativos. Los métodos enumerativos generan programaciones una a una utilizando procedimientos de eliminación inteligentes para verificar si la no optimalidad de una planificación implica la no óptima de muchas otras que aún no se han generado, lo que evita la necesidad de buscar en el espacio completo soluciones no viables.

7.1.2. Formulaciones matemáticas

Se ha reconocido que los problemas de programación de tareas se pueden resolver de manera óptima utilizando técnicas de programación matemática y una de las formas más comunes para el problema tipo Job Shop es el formato de programación lineal mixta (MIP) de Manne [16]. El formato MIP es simplemente el de un programa lineal con un conjunto de restricciones lineales y una única función objetivo lineal, pero con la restricción adicional de que algunas de las variables de decisión (y_{ipk}) son números enteros. Aquí las variables enteras son binarias y se utilizan para implementar las restricciones disyuntivas.

En Giffler y Thompson [17] se menciona que los programas de enteros no han dado lugar a métodos prácticos de soluciones, mientras que French [15] se expresa la opinión de que una formulación de programación de problemas enteros son inviables computacionalmente. Nemhauser y Wolsey [18] y Blazewicz [19] enfatizan aún más estas dificultades e indican que los modelos de programación matemática aún no han logrado el avance para los problemas de programación. Como resultado, estas técnicas solo son capaces de resolver instancias de juguete altamente simplificadas, en un tiempo razonable. Esto, como es lógico, sugiere que las técnicas adecuadas para Job Shop se encuentran en otros dominios.

En los métodos de Lagrangian relaxation LR, la prioridad y las restricciones de capacidad se relajan utilizando multiplicadores de Lagrangiano no negativos, con términos de penalización también incorporados en la función objetivo, mientras que la descomposición aborda el problema original en una serie de subproblemas más pequeños y manejables que luego se resuelven de manera óptima. Los resultados indican que los límites inferiores que generan no son muy buenos, pueden ser difíciles de calcular y, en algunos casos, debido al tiempo de computación excesivo requerido, la búsqueda debe terminarse en el nodo raíz. Es evidente que los enfoques matemáticos son inadecuados para Job Shop. En consecuencia, el enfoque principal de los enfoques enumerativos para el Job Shop son las técnicas de Branch & Bound.

7.1.3. Técnicas de Branch & Bound

Los algoritmos Branch & Bound (BB) usan una estructura de árbol construida dinámicamente como medio para representar el espacio de solución de todas las secuencias factibles. La búsqueda comienza en el nodo superior y se logra una selección completa una vez que el nodo de nivel más bajo ha sido evaluado.

Cada nodo en un nivel p en el árbol de búsqueda representa una secuencia parcial de operaciones p. Como lo indica su nombre, se aplica una bifurcación y un esquema de delimitación para realizar la búsqueda. Desde un nodo no seleccionado (activo), la operación de bifurcación determina el siguiente conjunto de nodos posibles a partir de los cuales podría progresar la búsqueda. Las dos estrategias de ramificación más comunes son la generación de horarios activos (GAS) y la resolución de conflictos esenciales (SEC) [20] [7]. GAS se deriva del trabajo de Giffler y Thompson [17]. Aquí, cada nodo consta de un programa parcial y el mecanismo de bifurcación arregla el conjunto de operaciones para secuenciar a continuación, mientras que en la bifurcación de SEC determina si O_i debe secuenciarse antes de O_j o viceversa. Barker y McMahon [7] indican que la SEC proporciona una mayor flexibilidad y, en general, se encuentra que es superior al GAS.

El procedimiento de delimitación selecciona la operación que continuará la búsqueda y se basa en un Lower Bound (LB) estimado y en el Upper Bound (UB) mejor logrado actualmente. En la mayoría de las técnicas de BB, el primer UB normalmente se proporciona por medio de una heurística y se aplica antes de que realmente comience la búsqueda de BB. Si en cualquier nodo el LB estimado es mayor que el mejor UB actual, no hay necesidad de continuar la búsqueda con esta selección parcial, ya que no puede mejorar el UB existente. Por lo tanto, la selección parcial y todos sus descendientes subsiguientes no se tienen en cuenta. Una vez que un nodo de hoja o un nodo en el que el LB es mayor que el mejor UB actual es descifrado, la búsqueda regresa (retrocede) al nodo no insinuado más alto del árbol. La búsqueda se detiene una vez que todos los nodos se han buscado

de forma implícita o explícita.

Los límites ajustados son, por lo tanto, esenciales para las técnicas de BB, ya que evitan la necesidad de buscar grandes secciones del espacio de la solución. Muchos tipos de límites se describen en la literatura. Aunque Akers [9], Brucker [21] y Brucker y Jurisch [22] han generado LB al reducir JSP en subproblemas de dimensionalidad $2xm$, $2xm$ y $3xm$ respectivamente, la formulación más popular es descomponer el conjunto de operaciones en m instancias de una máquina. El límite de máquina único se obtiene a partir de la fabricación del procesador de cuello de botella, que es el límite más fuerte en todas las máquinas. A pesar de que este problema es del tipo $NP - Hard$ [20] se han desarrollado buenas técnicas para su solución [23].

Además de las estrategias de bifurcación y delimitación, las reglas de inferencia o proposiciones, que intentan corregir el orden de varias operaciones, también forman parte integral de muchos algoritmos de BB. Al combinar con éxito estos tres componentes, se pueden eliminar grandes áreas del espacio de la solución en una etapa temprana de la búsqueda. La técnica de búsqueda de BB fue estudiada inicialmente por Brooks y White [24], Ignall y Schrage [25] y Lomnicki [26].

Balas [27] presenta una de las primeras aplicaciones de un esquema BB a JSP. Este método aplica el modelo de gráfico disyuntivo y solo considera operaciones críticas. Otro algoritmo basado en principios muy similares es el de Carlier [28] en el cual, un trabajo crítico, C y un conjunto crítico, J de operaciones, se derivan de esta secuencia y la dicotomía definida por la posición de C en relación con J sirve como base para la regla de ramificación.

Usando muchas de las reglas aplicadas por Carlier y Pinson [14], Applegate y Cook [12] determinan si una operación i no secuenciada debería secuenciarse antes o después de un conjunto fijo de operaciones, g. Esta estrategia se conoce como búsqueda de bordes, ya

que determina en qué borde de g, O_i se programará y se aplicará tanto en la ramificación como en la estrategia de delimitación.

Otro ejemplo de un trabajo derivado de Carlier y Pinson [29] donde la ramificación se basa en el esquema de bloque crítico de Grabowski et al. [30]. Perregaard y Clausen [31] modifican estas dos técnicas para permitir que se busque el espacio de la solución de manera más eficiente. El primer método proporciona una estrategia de búsqueda paralela para el algoritmo de Carlier y Pinson [32] utilizando un enfoque de balanceo de carga, mientras que el segundo método es una versión paralela del algoritmo de Brucker [29] y aplica un arreglo maestro/esclavo.

Si bien todos los métodos descritos hasta ahora aplican el modelo de gráfico disyuntivo, Martin [33] adopta una representación orientada en el tiempo a la variante de decisión de JSP. El procedimiento de delimitación más poderoso creado es una técnica llamada afeitado. A cada operación se le asigna una ventana de tiempo en la que se puede procesar y se basa en varias reglas y selecciones cada vez que se puede cumplir un tiempo objetivo T una o más unidades de tiempo se intenta eliminar (afeitar) de T, reduciendo las ventanas de tiempo de diversas operaciones. El objetivo es hacer que la ventana de tiempo de cada operación sea lo más pequeña posible, evitando conflictos de recursos.

7.2. Métodos de aproximación

Aunque los métodos de aproximación no garantizan el logro de soluciones exactas, pueden alcanzar soluciones casi óptimas en tiempos de computación moderados y, por lo tanto, son más adecuados para problemas más grandes. La importancia de los métodos de aproximación son indicadas por Glover y Greenberg [34], quienes sugieren que la búsqueda dirigida de árboles es totalmente insatisfactoria para problemas combinatorios difíciles. Indican que las heurísticas inspiradas por fenómenos naturales y la resolución inteligente de problemas son las más adecuadas para proporcionar un vínculo bilateral apropiado entre

la investigación de operaciones y la inteligencia artificial. En este análisis, se consideran cuatro categorías principales de técnica de aproximación: reglas de despacho de prioridad, heurísticas basadas en cuellos de botella, inteligencia artificial y métodos de búsqueda locales.

7.2.1. Reglas de Despacho Prioritario (PDRS)

Los procedimientos de aproximación aplicados a JSP se desarrollaron por primera vez sobre la base de las reglas de envío prioritarias y, debido a su facilidad de implementación y su requisito computacional sustancialmente reducido, son una técnica muy popular [7]. En cada paso sucesivo, a todas las operaciones que están disponibles para ser programadas se les asigna una prioridad y la operación con la prioridad más alta se elige para secuenciarse.

Por lo general, se realizan varias ejecuciones de PDRS para lograr resultados válidos, de particular interés es el algoritmo de Giffler y Thompson [17] considerado como la base común de todos los PDRS y su importancia se deriva del hecho de que genera horarios activos. El procedimiento comienza al elegir la operación sin secuencia, O_j, con el primer tiempo de finalización, luego encuentra todas las demás operaciones que usan la misma máquina y que comienzan antes que el tiempo de finalización de O_j. Estas operaciones se colocan en un conjunto de conflicto. A continuación, se selecciona una operación del conjunto de conflictos y se secuencia lo antes posible. El procedimiento se repite hasta que se hayan secuenciado todas las operaciones. Los PDRS se caracterizan por el método aplicado para seleccionar operaciones del conjunto de conflictos donde una elección aleatoria es la asignación de prioridad más simple.

Panwalker e Iskander [35] aportan un estudio completo completa de heurísticas de programación donde se presentan, revisan y clasifican 113 PDRS. Haupt [36] proporciona una discusión extensa y un resumen de estos y muchos otros PDRS. En el es estudio evalúan el desempeño de 42 PDRS usando un modelo de programación lineal. Su análisis indica que las reglas relacionadas con el tiempo de procesamiento más corto (SPT) se desempeñan

bien de manera consistente, mientras que las reglas basadas en el tiempo de procesamiento más largo (LPT) se comportan mal de manera consistente. Como las reglas individuales funcionan tan mal y no proporcionan una conclusión clara con respecto al criterio de makespan, se han formulado heurísticas más complejas. Por ejemplo, el algoritmo de Viviers [37] incorpora tres niveles de clases de prioridad dentro de la heurística SPT. El método más común para mejorar el rendimiento de la solución es tener una combinación probabilística de PDRS individuales.

Los primeros ejemplos de esta estrategia son los de Crowston [38] y Fisher y Thompson [13]. Lawrence [39] compara el rendimiento de diez reglas de envío de prioridad individuales con una combinación aleatoria de estas reglas y muestra que el método combinado proporciona resultados muy superiores pero que requiere un tiempo de computación sustancialmente mayor. Otros métodos más sofisticados utilizados para controlar la elección de qué regla aplicar incluyen un algoritmo genético y lógica difusa.

Es evidente que los PDRS simplemente eligen una operación posible para agregar a la secuencia parcial actual, mientras que las técnicas de Branch & Bound evalúan todas las operaciones posibles, ya sea de manera implícita o explícita. La técnica de búsqueda por haz [40] proporciona un equilibrio entre estos enfoques al evaluar varias soluciones óptimas en un punto de decisión determinado.

Otra técnica que incorpora la búsqueda por haz es el algoritmo de inserción laboral que consta de dos fases. La primera fase aplica un esquema de inserción para construir un programa. Una operación se coloca en el programa parcial de manera que minimiza el makespan que pasa a través de ella. La segunda fase aplica una estrategia de reinserción para mejorar de forma iterativa la solución inicial en la que los vecinos se eligen según el enfoque de bloque crítico. En ambas fases se aplica la búsqueda por haz para mejorar la búsqueda.

7.2.2. Heurística basada en cuellos de botella

Aunque durante muchos años los únicos métodos de aproximación viables fueron las reglas de envío prioritarias, el advenimiento de computadoras más potentes, así como el énfasis en técnicas cuidadosamente diseñadas, analizadas e implementadas [41] han permitido desarrollar enfoques más sofisticados que pueden acortar la brecha entre los PDRS y los métodos exactos de explosión combinatoria que consumen mucho tiempo.

Un ejemplo de tal enfoque es la heurística basada en cuellos de botella (SBP) de Adams [42]. SBP se caracteriza por las siguientes tareas: identificación de subproblema, selección de cuellos de botella, solución de subproblema y reoptimización de horarios. La estrategia real consiste en relajar el problema de JSP en los problemas de una máquina y resolver iterativamente cada subproblema uno por uno utilizando el enfoque de Carlier [28]. Cada solución de una máquina se compara con todas las demás y las máquinas se clasifican en función de su solución. La máquina sin secuenciar que tiene el mayor valor de solución se identifica como la máquina cuello de botella. SBP secuencia la máquina cuello de botella en función de las máquinas ya programadas, mientras que las máquinas sin secuenciar restantes se ignoran. La selección de la máquina de cuellos de botella está motivada por la conjetura de que programarla en una etapa posterior se deterioraría el makespan.

Cada vez que la máquina identificada como el cuello de botella está programada, todas las máquinas programadas previamente, susceptibles de mejora, se reoptimizan localmente resolviendo nuevamente el problema de la máquina. La principal contribución de este enfoque es la forma en que se utiliza la relajación de una máquina para decidir el orden en que deben programarse las máquinas. En 1996 se proporcionan un análisis computacional de los componentes individuales de la SBP, quienes indican que la calidad de la solución y el tiempo de computación se ven afectados significativamente por la estructura de enrutamiento.

Cuando SBP produce la secuencia en una máquina, puede crear restricciones de precedencia entre pares de trabajos en una máquina sin secuencia. Estas restricciones, conocidas como restricciones de precedencia demorada (DPC), surgen porque la secuenciación de una máquina determinada puede imponer condiciones en la secuencia de alguna otra máquina, del tipo que el trabajo i tiene que preceder al trabajo j al menos en un lapso de tiempo específico. Debido a esta dependencia del trabajo cuando la instancia se relaja a un problema de programación de una máquina, está menos restringida de lo que debería ser, por lo tanto, no se selecciona la verdadera máquina de cuellos de botella, no se calcula la mejor secuencia, la reoptimización no garantiza una disminución monotónica de el makespan y la solución final de SBP pueden ser inviables.

7.3. Inteligencia Artificial

La inteligencia artificial (IA) es el subcampo de la informática que se ocupa de la integración de la inteligencia biológica y la informática. Tiene orígenes fundamentales a partir de la comprensión biológica y utiliza principios en la naturaleza para encontrar soluciones. Usando este entendimiento natural, uno de los objetivos principales de la IA es hacer que las computadoras sean más útiles para resolver problemas. Aquí se analizan dos metodologías principales de IA: Criterios de satisfacción de restricciones y métodos de redes neuronales. Muchas otras técnicas de IA, como los sistemas expertos se han aplicado a JSP, sin embargo, su efecto ha sido limitado.

7.3.1. Satisfación de restricción (CS)

Las técnicas de satisfacción de restricción apuntan a reducir el tamaño efectivo del espacio de búsqueda aplicando restricciones que restringen el orden en el que se seleccionan las variables y la secuencia en la que se asignan los valores posibles a cada variable. Después de asignar un valor a una variable, se elimina cualquier inconsistencia que surja. El proceso de eliminar valores inconsistentes se llama verificación de consistencia, mientras que el método de deshacer las asignaciones anteriores se conoce como retroceso. Una

búsqueda de retroceso fija un orden en las variables y también determina un ordenamiento fijo de los valores de cada dominio. El problema de satisfacción de restricciones (CSP) se resuelve cuando se especifica una asignación completa de variables que no viola las restricciones del problema. Aunque se consideran dentro del dominio de AI, muchos métodos de programación basados en restricciones aplican una búsqueda sistemática en el árbol y tienen vínculos cercanos con los algoritmos de BB.

Los ejemplos de los primeros sistemas de programación basados en restricciones incluyen un esquema de programación de trenes por Fukumori [43]. Fox [44] diseña y construye un modelo de búsqueda heurística dirigida por restricción interactiva comocido como sistema de programación e información inteligente (ISIS), en el que se utilizan restricciones para vincular, guiar y analizar el proceso de programación.

Además de la construcción de los sistemas de JSP industriales, también se ha trabajado mucho en la aplicación de procedimientos de satisfacción de restricciones para resolver el modelo estándar de JSP. La mayoría del trabajo más reciente sobre métodos de satisfacción de restricciones se ha concentrado en un pequeño grupo de investigadores. En uno de sus primeros trabajos, Fox y Sadeh [45] ofrecen una serie de enfoques de satisfacción de restricciones para problemas de programación cada vez más difíciles que implementan un enfoque que involucra la programación de operaciones con ventanas de tiempo de inicio más antiguas/más recientes predefinidas.

Sadeh y Fox [45] aplican un marco que asigna una probabilidad subjetiva, basada en la contención de recursos, a cada operación no programada. Estos autores también presentan una serie de algoritmos de optimización y aproximación basados en restricciones. Nuijten y Aarts [46] adaptan muchos de estos métodos al problema de programación de múltiples talleres capacitados (MCJSP). Otros trabajos recientes introducen el concepto de intervalos de tareas que consiste en todas las operaciones que pueden programarse entre la hora de inicio más temprana de i y la hora de finalización más reciente de j.

Las propuestas de búsqueda de bordes de Applegate y Cook [47] son entonces aplicadas para determinar el orden de operación donde formulan un modelo híbrido de dos etapas. El primer paso permuta las disyunciones en la ruta crítica, mientras que el segundo paso corrige una pequeña parte de la solución y luego aplica el algoritmo del buscador de bordes con intervalos de tareas para completar el resto del programa. Harvey y Ginsberg [48] presentan un método llamado Búsqueda de discrepancia limitada (LDS) que intenta eliminar las decisiones erróneas tomadas al principio de la búsqueda, buscando sistemáticamente todas las posibilidades que difieren de la posibilidad elegida en un número pequeño de puntos de decisión.

7.3.2. Redes neuronales (NNs)

Las redes neuronales (NN) se organizan en un marco basado en la estructura cerebral de entidades vivientes simples. En estas técnicas, el procesamiento de la información se lleva a cabo a través de una red masivamente interconectada de unidades de procesamiento paralelo. Su simplicidad, junto con su capacidad para realizar computación distribuida, así como su propensión a aprender y generalizar, ha hecho de las redes neuronales una metodología popular, lo que les permite ser utilizados en muchas aplicaciones de la vida real. Cheung [49] describe algunas de las principales arquitecturas de redes neuronales aplicadas para resolver problemas de planificación: red de búsqueda (red Hopfield), red de corrección de errores (Perceptrón multicapa), red probabilística (máquina de Boltzmann), red competidora y red autoorganizada. Sin embargo, como las aplicaciones de red neuronal para JSP se han implementado principalmente en los dos primeros paradigmas, el enfoque aquí se centra únicamente en estos dos modelos.

Las redes de búsqueda, como las redes Hopfield, son redes no lineales autoasociativas, que tienen una dinámica inherente para minimizar la función de energía del sistema o la función Lyapunov. En muchos de los métodos basados en Hopfield, se aplica un modelo matemático subyacente para mapear JSP en la red neuronal. El más común es el modelo

de programación lineal mixta (MIP) de Manne [16].

Las redes de corrección de errores, por otro lado, están entrenadas en ejemplos que toman la forma de un mapeo desde algún subconjunto delimitado arbitrario del espacio euclidiano n-dimensional al espacio euclidiano m dimensional. Cuando se aplica un patrón de actividad a la red, la regla de corrección de errores ajusta los pesos sinápticos de acuerdo con la asignación anterior. Específicamente, la respuesta real de la red se resta de la respuesta objetivo deseada para producir la señal de error. Los pesos se ajustan para que la respuesta real de la red se acerque más a la respuesta deseada.

7.3.3. Redes de Hopfield

La Red Hopfield [50] domina los sistemas de programación basados en redes neuronales y cuando se aplica a JSP, el objetivo es minimizar la función de energía, E, que se basa en makespan, sujeta a las diversas restricciones de prioridad y recursos. Si se violan las restricciones, se produce un valor de penalización que incrementa E. En una de las primeras obras, Foo y Takefuji [51] utilizan una codificación similar a la formulación TSP de Hopfield y Tank [50] para asignar JSP a un matriz bidimensional mn por (mn + 1) neuronas.

Para mejorar su método anterior, Foo y Takefuji [51] construyen una red neuronal de programación lineal de enteros (ILPNN) mediante la formulación de JSP como un problema MIP. La función de energía está representada por la suma de los tiempos de inicio de todos los trabajos y las soluciones se obtienen realizando ajustes de programa lineal y entero (binario) hasta la convergencia.

Van Hulle [52] indica que el enfoque de Foo y Takefuji [51] no garantiza soluciones factibles. Como resultado, traduce la formulación MIP a un formato de programación de objetivos que luego se asigna a una red neuronal (programación de objetivos). Willems y Rooda [51] también usan una formulación MIP, pero intentan superar algunas de las

dificultades anteriores reduciendo el espacio de búsqueda de su ILPNN a través del cálculo previo. Para superar las limitaciones que enfrentan los sistemas ILPNN, se propone una construcción de la Red Neuronal de Programación Lineal (LPNN) evitando el uso de una función de energía cuadrática al implementar una función lineal en su lugar; esto evita la necesidad de un método de programación lineal de enteros convencional que requiere variables de control excesivas. Cherkassky y Zhou [52] comparan este modelo neural con tres PDRS de uso común. Los resultados muestran que, excepto en un caso, la red neuronal funciona mejor.

7.3.4. Enfoques diversos

Los ejemplos de enfoques diversos de IA incluyen un modelo estocástico distribuido paralelo de Lo y Hsu [53] que tiene muchas similitudes con la red neuronal de Hopfield estocástica. Un método de potencial de vibración basado en analogías de la dinámica y la física estadística donde se observa el potencial físico de cada trabajo como la energía de interacción entre las operaciones en las máquinas y un método de optimización llamado Ant System (AS) para JSP por Colorni [54]. AS es una técnica de optimización estocástica basada en la población desarrollada por Denebourg, Pasteels y Verhaeghe [55] donde se modela el comportamiento que tienen las colonias de hormigas para encontrar el camino más corto hacia su fuente de alimento.

La hormiga es un agente de cooperación simple cuyo rendimiento de búsqueda se vuelve muy efectivo cuando se trabaja en conjunto con muchos otros agentes simples, lo que permite trascender los mínimos locales pobres. Las hormigas se colocan al azar en los nodos de un gráfico disyuntivo y siguen un procedimiento de Monte Carlo combinado con una heurística golosa para decidir a qué nodo adyacente se moverá. Los resultados indican que los métodos de IA generalmente han tenido un bajo rendimiento, mientras que el esfuerzo computacional requerido es alto. En contraste, el grupo final de técnicas de aproximación que se analizarán: los métodos de búsqueda local han logrado un mayor grado de éxito.

7.4. Métodos de búsqueda local y metaheurísticas.

Para obtener una solución algorítmica para un determinado problema de optimización combinatoria P, donde R es el conjunto de soluciones factibles para P, a menudo es necesario definir configuraciones, es decir, un conjunto finito de soluciones, una función de costo a optimizar y una generación. Mecanismo, que es una prescripción simple para generar una transición de una configuración a otra mediante un pequeño cambio. Tales métodos se conocen como búsqueda local o técnicas de búsqueda por vecindario.

En la búsqueda local, el mecanismo de generación describe un vecindario para cada configuración. Un vecindario es una función que define una transición simple de una solución x a otra solución al inducir un cambio que normalmente se puede ver como una pequeña perturbación. Cada solución x se puede alcanzar directamente de x por una sola transformación parcial predefinida de x llamada movimiento, (el término solución se concibe en el sentido de uno que satisface ciertos requisitos estructurales de un problema en cuestión, pero no es necesario que todos los requisitos son factibles.)

El objetivo de estas estrategias es perturbar progresivamente la configuración actual a través de una sucesión de vecinos para dirigir la búsqueda hacia una solución mejorada. La mejora se busca en cada paso mediante métodos de ascenso estándar, o en algunos (posiblemente) mayor número de pasos mediante métodos más avanzados. En los casos en que las soluciones pueden involucrar infactibilidades, la mejora a menudo se define en relación con un objetivo modificado que penaliza tales infactibilidades. En estos métodos, la selección de un vecino está dictada por los criterios de elección. Varios procedimientos de selección comunes incluyen:

- Elegir el primer vecino de menor costo encontrado: esto se conoce como primera mejora y se aplica en los algoritmos de umbral.
- Seleccionar el mejor vecino de todo el vecindario: esto se conoce como mejor mejora y se realiza mediante el procedimiento de cuello de botella cambiante y los algoritmos

de inserción.

Desde una perspectiva general, la solución a JSP se puede considerar como un conjunto de decisiones locales con respecto a qué operación programar a continuación. Dorndorf y Pesch [56] sugieren que se debe construir un marco que navegue estas decisiones locales a través del dominio de búsqueda para determinar una solución global de alta calidad en un tiempo razonable. En un marco de este tipo, las decisiones locales tomadas por heurísticas específicas del problema miope se guían más allá de la optimalidad local mediante una metaestrategia subyacente. Esto da lugar a algoritmos de búsqueda local iterados o metaheurísticas, que combinan las propiedades específicas del problema de la búsqueda local con las propiedades genéricas del metasolver, lo que permite lograr buenas soluciones.

En Evans [57], uno de los primeros trabajos analíticos, se investiga la efectividad de los mecanismos de generación de búsquedas locales desde la perspectiva del estado-espacio de la inteligencia artificial. Además de revisiones analíticas, también se han realizado estudios experimentales. El principal tipo de análisis realizado en algoritmos de búsqueda local se refiere a problemas de complejidad. Los principales trabajos en esta área son de Johnson [6] que definen la clase de complejidad conocida como polinomial-time local search (PLS) que establecen un marco formal con respecto a la teoría de la complejidad de la búsqueda local, así como la definición más clara de los problemas de Job Shop de complejidad asociados.

Las técnicas metaheurísticas son el desarrollo más reciente en los métodos de búsqueda aproximados para resolver problemas complejos de optimización. Las metaheurísticas JSP se basan en las estrategias de vecindad desarrolladas por Matsuo [58].

7.5. Métodos basados en el espacio de soluciones del problema

Los métodos basados en el espacio de soluciones de problemas son heurísticas de dos niveles que generan muchas soluciones de inicio diferentes, utilizando técnicas constructivas

rápidas específicas del problema, que luego se mejoran mediante la búsqueda local. Los ejemplos incluyen la búsqueda en el espacio problema y en el espacio heurístico y los codiciosos procedimientos de búsqueda adaptativa aleatoria.

7.5.1. Búsqueda en el espacio problema y en el espacio heurístico

En [59] se presenta la idea que muchos métodos solo abordan el problema de cómo buscar el espacio de la solución, no dónde buscar. En consecuencia, definen los vecindarios de búsqueda en el espacio problema y en el espacio heurístico en lugar de en el espacio de solución más tradicional, que se basa en operaciones de intercambio. Ambas definiciones aplican una heurística de base rápida, h, para generar un vecindario apropiado de soluciones. En el problema, el espacio h se aplica a las versiones perturbadas del problema original, mientras que en el espacio heurístico la búsqueda se basa en la capacidad de definir versiones parametrizadas de h. Luego se utiliza un algoritmo de mejora iterativo simple para realizar la búsqueda de vecindario.

Los resultados indican que el algoritmo genético del espacio problema es claramente la mejor técnica. Los buenos resultados del enfoque genético se deben al hecho de que es relativamente fácil de codificar en el espacio heurístico y en el espacio de problemas en comparación con el recocido simulado y la búsqueda tabú. Como resultado, se concluye que los métodos de recocido simulado y búsqueda tabú carecen de sofisticación.

7.5.2. Procedimiento de búsqueda adaptativa aleatoria golosa (GRASP)

El procedimiento de búsqueda adaptativa aleatoria golosa (GRASP) es un método basado en el espacio de problemas que consiste en una fase constructiva y otra iterativa. En la fase de construcción, la solución se construye un elemento a la vez. Todos los elementos posibles que se pueden elegir a continuación están ordenados en una lista de candidatos con respecto a una función golosa. Algunos de los mejores candidatos se colocan en una lista de candidatos restringida (RCL). La naturaleza adaptativa de GRASP se deriva de

su capacidad para actualizar los valores asociados con cada elemento, en cada iteración, según la selección que se acaba de realizar. Mientras que la naturaleza probabilística del algoritmo se deriva de la selección aleatoria de un elemento en el RCL.

En la fase iterativa se aplica un procedimiento de búsqueda local, que reemplaza sucesivamente la solución actual por una mejor en su vecindario. Esta fase se detiene cuando no se puede encontrar un mejor vecino. Luego, el proceso vuelve a la fase de construcción y se crea una nueva solución inicial. El algoritmo termina una vez que se alcanza la solución general deseada o un número predeterminado de iteraciones, y devuelve la mejor solución encontrada hasta el momento. Resende [60] presenta una aplicación de GRASP a JSP. El RCL consiste en las operaciones que, cuando se secuencian a continuación, darían como resultado el tiempo de finalización general más bajo de la secuencia parcial.

7.6. Algoritmos del umbral

Uno de los grupos más populares de métodos de búsqueda local iterados son los algoritmos de umbral, que eligen una nueva configuración si la diferencia en el costo entre el vecino y la solución actual está por debajo de un umbral dado (L).

7.6.1. Mejora iterativa (IM)

El ejemplo más simple de un algoritmo de umbral es el de la mejora iterativa donde los umbrales se establecen en 0, por lo tanto, solo se aceptan mejores configuraciones. Desde un programa inicial generado aleatoriamente, estos métodos dirigen la búsqueda a un óptimo local, ya que la solución es al menos tan buena o mejor que todas las soluciones en su vecindario. Una vez que alcanza el óptimo local, ya que no aceptará movimientos que no mejoren, queda atrapado. La IM es la clase más simple de técnica de búsqueda local iterada, que forma la base de otros métodos más elaborados. Mientras que IM acepta la primera solución de mejora en su vecindario, una variación simple de esto, conocida como descenso más pronunciado, evalúa todos los movimientos en su vecindad y selecciona la

que proporciona la mejoría.

Se aplica un algoritmo de mejora iterativo de inicio múltiple (MSIM) que termina cuando se alcanza un límite en el tiempo total de ejecución. Este límite es el mismo para todos los algoritmos evaluados en su estudio computacional. Su estudio compara MSIM, recocido simulado (SA), umbral de aceptación (TA) y búsqueda local genética (GLS) cuando se aplica con los vecindarios de Matsuo [58]. El método MSIM comienza a partir de una secuencia generada aleatoriamente. El algoritmo luego mejora iterativamente la solución actual con una mejor en su vecindario. Una vez que no se puede encontrar un mejor vecino, es decir, se logra un óptimo local, la búsqueda se reinicia desde otro punto de inicio elegido al azar. A partir de lo cual se determina un óptimo local. Este proceso se repite hasta que se cumplen los criterios de terminación y se devuelve la mejor solución encontrada.

7.6.2. Optimización de gran paso

La optimización de grandes pasos, desarrollada por Martin, Otto y Felten [61] indican que es una subclase de enfoques influyentes de diversidad y perturbación, ya que impulsa una secuencia de óptimos locales a una distancia mayor que la habitual desde su ubicación actual, hasta soluciones de buena calidad. Es una técnica relativamente nueva con una aplicación limitada a JSP.

Las pruebas realizadas por Lourenço [62] indica que las SA son las que mejor realizan los pasos pequeños de SA e IM, sin embargo, SA lleva más tiempo que la IM. De las técnicas de grandes pasos analizadas, los mejores métodos son las dos implementaciones de máquinas aleatorias. Uno basado en el método especificado en Carlier [28] (2rand-car) y el otro aplicando la Regla de fecha de vencimiento más antigua (2rand-edd). Los resultados indican que los pasos grandes con recocido simulado son mejores que el recocido simulado solo, sin embargo esta diferencia se hace más pequeña a medida que aumenta el tiempo permitido para ambos métodos.

En un trabajo reciente [62], los pasos pequeños se realizan utilizando la búsqueda tabú y los pasos grandes se aplican utilizando la técnica de $2rand - car$. Los resultados con criterios de terminación basados en iteración indican que la optimización de grandes pasos con la búsqueda tabú se realiza mejor que la búsqueda tabú, que a su vez es mejor que la optimización de grandes pasos mediante el recocido simulado.

7.6.3. Simulated annealing (SA)

Simulated annealing (SA) es una técnica de búsqueda orientada al azar que se introdujo como una analogía de la física estadística de la simulación por computadora del proceso de recocido de un metal caliente hasta que se alcanza su estado de energía mínima. Se basa en las propuestas independientes de Kirkpatrick [63] y Cerny [64]. En SA, las configuraciones son análogas a los estados de un sólido, mientras que la función de costo f y el parámetro de control c son equivalentes a la energía y la temperatura, respectivamente.

7.7. Algoritmos genéticos (GA)

La arquitectura del algoritmo genético (GA) fue propuesta por primera vez por el profesor John Holland [65] donde la principal inspiración es el mecanismo evolutivo en la biosfera. El GA se basa en un modelo abstracto de evolución natural, de manera que la calidad de los individuos se construye al más alto nivel compatible con el medio ambiente. El análisis que se presenta aquí se basa libremente en la clasificación de todos los esquemas de representación aplicados en los últimos años en 9 categorías:

1. Basada en la operación
2. Basado en el trabajo
3. Relación de par de trabajos basada
4. Tiempo de finalización

5. Claves aleatorias

6. Lista de preferencias basada

7. Regla de prioridad basada

8. Basado en el gráfico disyuntivo

9. Basada en la máquina

Estos esquemas de representación se pueden agrupar en dos enfoques de codificación básicos, directos e indirectos. El enfoque directo codifica un programa de JSP como un cromosoma y los operadores genéticos se utilizan para hacer que estos cromosomas se conviertan en mejores programas. Las categorías 1 a 5 son ejemplos de esta estrategia. En el enfoque indirecto, una secuencia de preferencias de decisión se codifica en el cromosoma, por ejemplo, las reglas de envío para las asignaciones de trabajo, y los operadores genéticos se aplican para mejorar la Ordenación de las distintas preferencias. Luego se genera un programa de JSP a partir de la secuencia de preferencias.

El método de GA más antiguo aplicado a JSP es de Davis [66], que es un ejemplo de un enfoque indirecto. Su técnica construye un ordenamiento preferido de operaciones para cada máquina. Falkenauer y Bouffouix [67] amplían aún más este enfoque, que codifica las operaciones que se procesarán en una máquina como una lista de preferencias que consiste en una cadena de símbolos. Della Croce et al. [68].

Una de las representaciones basadas en listas de preferencias más recientes es la de Kobayashi [69] donde un cromosoma es una cadena de símbolos de longitud n y cada símbolo identifica la operación a procesar en una máquina. Tamaki y Nishikawa [70] aplican una indirecta representación basada en el gráfico disyuntivo donde Un cromosoma consiste en una cadena binaria que corresponde a una lista de preferencias ordenadas de bordes disyuntivos.

Uno de los primeros enfoques directos es de Nakano y Yamada [71] quienes crean una codificación binaria basada en la relación de precedencia de las operaciones en la misma máquina. También se adopta una estrategia llamada forcing que modifica un cromosoma si una operación se puede desplazar a la izquierda. Yamada y Nakano mejoran este trabajo al definir un operador de cruce llamado GA/GT sobre el algoritmo de Giffler y Thompson [17], aquí el cromosoma es una lista ordenada de los tiempos de finalización de las operaciones.

Bierwirth [72] crea un algoritmo genético de permutación generalizada para mejorar los métodos existentes. Un cromosoma representa una permutación de trabajos. En se analizan tres operadores de cruce, que preservan, respectivamente, el orden relativo, de posición y de permutación absoluta de las operaciones. Mattfeld [73] proporciona una investigación detallada de la aplicación de técnicas evolutivas a JSP.

Yamada y Nakano [74] generan dos horarios principales, lo más alejados posible con respecto a su distancia gráfica disyuntiva. Al iniciar la búsqueda en el primer padre, reemplazan de forma iterativa una solución en la población actual con una secuencia mejorada orientada hacia el segundo padre. Realizaron mejoras adicionales a este método, donde se aplica un esquema de reemplazo estocástico sesgado que favorece las soluciones que están más cerca del segundo padre y se aplica una estrategia de vecindario de bloque crítico. Tenga en cuenta que ambos métodos se basan en la idea de volver a vincular la ruta en la búsqueda tabú.

7.8. Búsqueda Tabú (TS)

La técnica de optimización iterativa global Tabu Search (TS) proviene de principios generales de resolución inteligente de problemas y se deriva de los trabajos de Glover [34]. En esencia, TS es un procedimiento de búsqueda orientado determinista simple que restringe la búsqueda y busca trascender la optimalidad local almacenando el historial de

búsqueda en su memoria. Prohíbe (hace tabú) movimientos en el vecindario que tiene ciertos atributos, con el objetivo de guiar el proceso de búsqueda fuera de las soluciones que (según la información disponible) parecen duplicar o parecerse a las soluciones alcanzadas anteriormente. La función de memoria a corto plazo permite el olvido estratégico al hacer solo el tabú más reciente de t movimientos. Sin embargo, el estado tabú de un movimiento no es absoluto. Los criterios de aspiración permiten seleccionar un movimiento tabú si alcanza un nivel determinado de calidad.

Las funciones de memoria a medio y largo plazo también pueden aplicarse para proporcionar una exploración más amplia del espacio de búsqueda. Las estrategias a medio plazo o intermedias se basan en modificar las reglas de elección para alentar movimientos y soluciones históricamente bien encontradas donde estos esquemas generalmente regresan a partes atractivas del dominio de búsqueda e intensifican la búsqueda en estas regiones. Varios enfoques de búsqueda tabú también se han aplicado a las generalizaciones de JSP.

8. Estrategia de solución

Job-shop Scheduling (JSP) es uno de los problemas de optimización del tipo extremadamente difícil o $NP - Hard$ debido a que requiere un espacio de búsqueda combinatoria muy grande junto con la restricción de precedencia entre máquinas. El algoritmo tradicional utilizado para resolver el problema es el método de Branch & Bound el cual demanda un tiempo de cálculo considerable cuando el tamaño del problema es grande. En este trabajo, se propone el uso de un algoritmo genético (GA) para resolver JSP utilizando un método de codificación que ha demostrado ser computacionalmente viable [75].

El Algoritmo genético se basa en el sistema de evolución y hereditario [65]. En el proceso de evolución la población se somete a recombinación sexual (crossover), mutación y selección natural que seleccionan los mejores individuos para la próxima generación, y así sucesivamente. En términos simples, los algoritmos genéticos se desarrollan a través del concepto de expresar una posible solución o parámetro de un problema en un cromosoma, codificando el cromosoma en una cadena o forma numérica donde cada valor de la cadena representa el gen en el cromosoma, indicando una parte de la solución, luego a través de la mutación y el apareamiento, se produce la próxima generación, es decir, diferentes soluciones potenciales y, finalmente, el concepto de supervivencia del más apto, eliminación de molestias, buena solución. Se hace una reserva para hacer la próxima ronda de mutaciones de apareamiento para producir una mejor solución hasta que se alcance la condición de detención establecida, esperando poder saltar de la solución local en el futuro para encontrar la solución óptima global.

Los componentes de GA son los siguientes:

1. Una representación simbólica de una solución
2. Una función de evaluación de soluciones de calificación en términos de su estado físico

3. Operadores genéticos que alteran la composición de crías mediante reproducción y recombinación

4. Una forma de crear una población inicial de soluciones

5. Valores de los parámetros en GA

En la figura 3, se ilustran cada uno de los componentes tenidos en cuenta en el presente trabajo.

Figura 3: Componentes del algoritmo genético

Este algoritmo se puede utilizar para resolver la mayoría de los problemas de optimización y los buenos resultados que genera son la razón para ser el método escogido para resolver el problema JSP.

Para explicar la metodología usada, se presenta un problema de JSP de $3x3$ con 3 piezas de trabajo y 3 máquinas. Cada pieza de trabajo tiene un orden de procesamiento diferente en cada máquina y cada pieza de trabajo pasa por 3 operaciones de mecanizado donde el objetivo de la programación es minimizar el tiempo total de finalización (Makespan). La tabla 1 registra la secuencia de procesamiento para cada pieza de trabajo en cada operación de mecanizado y la tabla 2 el tiempo de procesamiento requerido.

Secuencia de máquinas			
Operación	O_1	O_2	O_3
Job 1	M2	M3	M1
Job 2	M1	M3	M2
Job 3	M3	M2	M1

Tabla 1: Secuencia de máquinas

Tiempo de procesamiento			
Operación	O_1	O_2	O_4
Job 1	8	10	5
Job 2	15	9	12
Job 3	9	7	19

Tabla 2: Tiempos de procesamiento

8.1. Codificación y decodificación

Hay dos enfoques diferentes utilizados para abordar el problema de representación: la representación directa en el gen y la representación indirecta. En la representación directa, el programa en sí se utiliza como un individuo. Por lo tanto, no es necesario un procedimiento de decodificación, sino que requiere operadores de recombinación complicados que deberían garantizar que los individuos sean factibles. Si se asume la carga de desarrollar operadores genéticos complicados, este esquema tiene la ventaja de ser muy directo. La representación indirecta necesita un programa de interpretación para traducirlo a un horario válido. La ventaja de este esquema es la simplicidad de la estructura individual y los operadores, y el inconveniente es que el algoritmo genético está obligado a buscar solo en el espacio de todas las posibles permutaciones de los genes individuales.

Se adopta el enfoque indirecto para representar al individuo. La representación se puede ver como una especie de lista de trabajos ampliada, que contiene genes NxM, donde N es el número de trabajos y M es el número de máquinas. Debido a que se supone que cada trabajo se procesa en cada máquina solo una vez, cada trabajo aparece en la lista de trabajos exactamente M veces. Como una programación válida, se da el orden de procesamiento de los trabajos para cada máquina y el individuo indirecto se puede traducir a una programación válida de acuerdo con el orden predeterminado de procesamiento de las máquinas.

La referencia principal en este trabajo es el método de codificación de cromosomas en JSP propuesto por Gen-Tsujimura-Kubota [75]. El concepto de este método es representar el cromosoma como la secuencia de procesamiento de un conjunto de piezas de trabajo. Un gen representa el procesamiento de una pieza de trabajo. De acuerdo con la cantidad de veces que aparece la pieza de trabajo en el cromosoma, se conoce el procesamiento actual de cada pieza de trabajo.

Los cromosomas del ejemplo propuesto para explicar la metodología contarán con 9 genes cada uno debido a que es un problema $3x3$ con 3 máquinas y 3 trabajos. La codificación usada se representa en la siguiente figura 4.

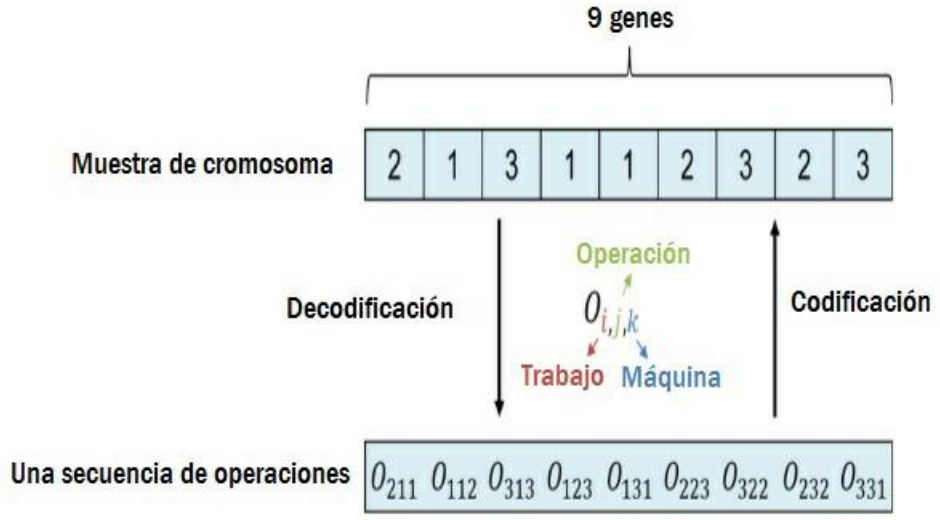

Figura 4: Representación de la codificación y decodificación del gen

8.2. Generar una solución inicial

Al completarse el diseño del cromosoma comenzará el cuerpo formal del algoritmo. Se debe generar un grupo de cromosomas como grupo inicial, conocido como solución inicial o población inicial. Los padres iniciales serán conocidos como la primera generación de antepasados de un organismo, y luego a través del apareamiento y la mutación, se producirá descendencia para criar más y mejores soluciones. En este caso, la población inicial se genera de manera aleatoria.

En los GA, las llamadas operaciones genéticas (apareamiento y mutación) se utilizan para generar descendencia, es decir, para generar nuevas soluciones potenciales (por supuesto, también es posible generar soluciones repetitivas), y se espera que se pueda lograr la exploración. El efecto, que aumenta la diversidad de soluciones, espera separarse de las

soluciones locales y encontrar más y mejores soluciones.

8.3. Apareamiento

Debido a que la representación no es la permutación de los trabajos, los bien conocidos operadores cruzados como Parcially Mapped Crossover (PMX), Cycle Crossover(CX), Order Crossover (OX) o Edge Recombination (ER) no son adecuados debido a la compleja relación de mapeo. Se propone un cronograma parcial de intercambio cruzado. Primero, los horarios parciales se seleccionan al azar en los padres respectivamente. El cronograma parcial se identifica con el mismo trabajo en las primeras y últimas posiciones del cronograma y luego se intercambiar los dos horarios parciales. Por lo general, los programas parciales tienen una longitud diferente (contiene un número diferente de genes). Los descendientes generados después del intercambio pueden ser mayores o menores que la longitud determinada, por lo tanto es necesario eliminar genes relativamente excesivos o agregar genes relativamente necesarios para que se conviertan en descendientes factibles. El apareamiento se representa en la figura 5.

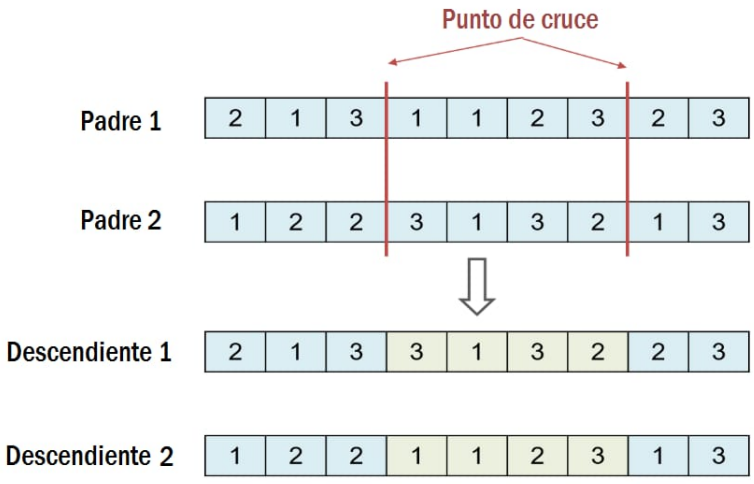

Figura 5: Cruce genético

8.3.1. Reparar

Este ejemplo es un problema JSP $3x3$ (3 trabajos y 3 máquinas) por lo que cada gen aparece en el cromosoma 3 veces, pero debido a la acción de apareamiento anterior, el número de genes en algunos cromosomas será menor que 10 o mayor que 9, formando una solución de programación inviable, por lo que es necesario reparar los cromosomas inviables para que sea un horario factible. La reparación de los genes se representa en la figura 6.

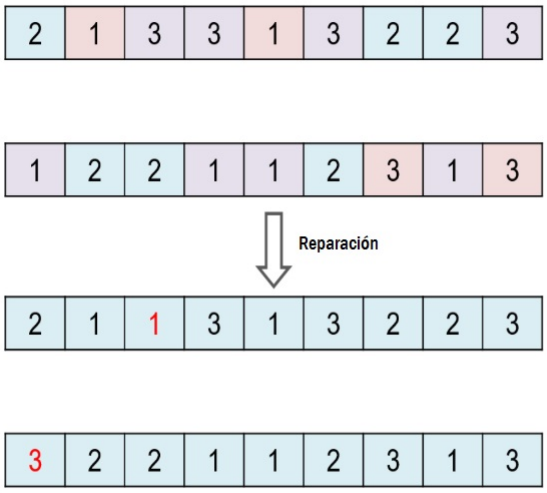

Figura 6: Reparación del cromosoma

8.4. Mutación

Aunque el apareamiento intenta mejorar la aptitud de sus descendientes, existen posibilidades de que estos converjan en una solución local. Al reparar esta situación, existe un mecanismo de escape llamado mutación, que cambia los genes de la misma lista al azar con poca probabilidad. Se define un operador de mutación de la siguiente manera: genera aleatoriamente dos posiciones en la lista e intercambia sus genes, y si los dos genes son el mismo, se vuelve a intentar seleccionar nuevas posiciones.

Con el fin de aumentar la diversidad de la solución y evitar caer en la solución local, se determina para cada cromosoma, si debe ser mutado de acuerdo a la tasa de mutación establecida. De ser así, el gen en un solo cromosoma se cambia de forma aleatoria. El método consiste en intercambiar varios genes en un cromosoma seleccionado al azar, como se muestra en la siguiente figura 7:

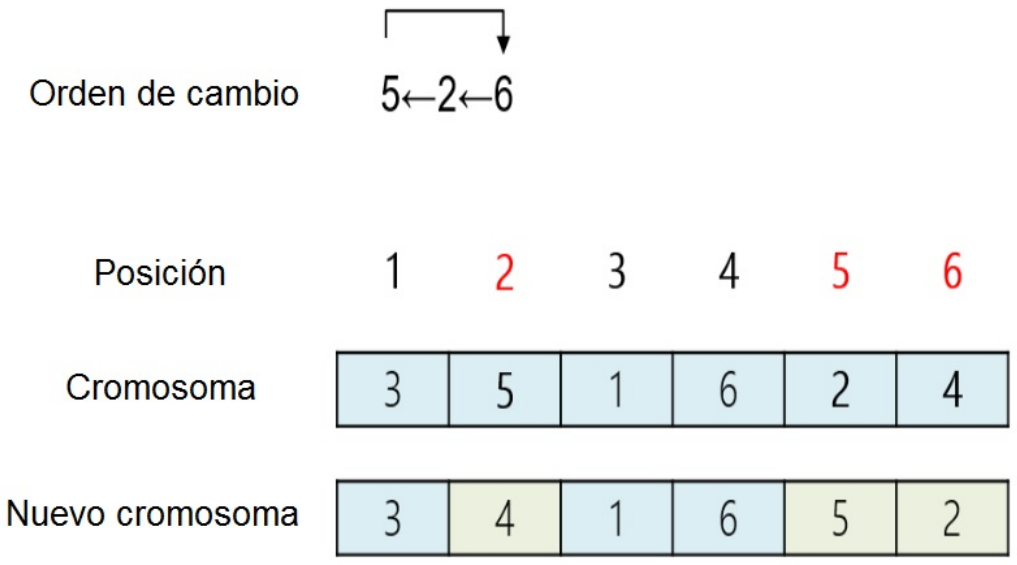

Figura 7: Mutación del cromosoma

8.5. Cálculo de la función adaptativa o fitness

Al resolver problemas con GA, se debe formular una función de aptitud física que se utilice para evaluar su calidad cromosómica. La función adaptativa se usa para evaluar dicha calidad [76]. El valor de aptitud del cromosoma se juzga por el valor de aptitud convertido. Cuanto mejor sea el valor de aptitud física, más probable será que el cromosoma se retenga para continuar multiplicándose cuando se seleccione el cromosoma en el

siguiente paso. Por el contrario, cuanto peor sea el valor de aptitud física, es más probable que se elimine. En general, la función adaptativa suele ser la función objetivo para resolver el problema, o una función suficiente para representar el problema para poder evaluar completamente la calidad del cromosoma. Básicamente, antes de realizar este paso, el cromosoma debe decodificarse para calcular aún más el valor de aptitud.

Durante cada iteración, los individuos en la población actual son evaluados, utilizando alguna medida de aptitud. Hay una serie de características de la función de evaluación que mejoran u obstaculizan el rendimiento de un programa de evolución. En la mayoría de las aplicaciones de optimización, la aptitud se calcula de acuerdo a la función objetivo natural. En este caso, la aptitud de cada individuo se evalúa como el tiempo total transcurrido del horario correspondiente.

Cabe señalar que, dado que se trata de un problema de minimización, el valor de aptitud calculado para cada cromosoma, es decir, el tiempo de finalización, debe registrarse de manera recíproca, de modo que cuando el método de la ruleta se use más tarde, seleccione el cromosoma con el menor tiempo de finalización, pero todavía hay otro registro del tiempo de finalización de cada cromosoma, de modo que la solución final se pueda comparar directamente con la mejor solución.

8.6. Selección

Para preservar mejores cromosomas para la evolución, debe existir un paso dentro del algoritmo que tenga en cuenta los cromosomas producidos en los pasos anteriores con el objetivo de seleccionar los cromosomas de mejor calidad para formar un nuevo grupo étnico. En este caso, el método de selección usado fue el mecanismo conocido como tipo ruleta.

8.6.1. Selección tipo ruleta

Los padres se seleccionan de acuerdo a su fitness. Los individuos mejores (con mayor fitness) son los que tienen mayores posibilidades de ser elegidos. Intuitivamente el proceso construye una ruleta o una tarta en la que cada una de las porciones representa a un individuo. La porción de tarta que le toca a cada individuo es proporcional a su fitness. Así, los individuos buenos se llevarán las mayores porciones y al revés ocurrirá con los peores. Ahora, al igual que en un casino, se lanza a la ruleta una canica. El lugar donde pare dicha canica, será ocupado por un cromosoma que será elegido. Resulta claro que los individuos con mayor fitness son los que más a menudo son elegidos.

Existe un algoritmo para realizar este proceso, el cual se presenta a continuación:

1. Calcular la suma total acumulada de los fitness de todos los individuos de la población actual.

2. Generar un número aleatorio entre 0 y la suma total acumulada conocido como r.

3. Recorrer la población acumulando nuevamente los fitness. Cuando la suma que se lleve sea mayor o igual a r, seleccionar el individuo donde se vaya recorriendo.

8.7. Conservar el mejor individuo

En cada generación, se presume que la función de aptitud de la población mejora en promedio. Sin embargo, los operadores genéticos pueden destruir al mejor individuo que apareció hasta el momento. Para eliminar este inconveniente, el mejor individuo se mantiene en cada generación, es decir, todos los individuos se comparan con el mejor que se había mantenido en la generación anterior, y si un individuo es mejor que el mejor individuo actual, dicho individuo se reemplaza como el nuevo mejor.

8.7.1. Comparación

Primero se compara el tiempo de finalización de cada cromosoma, se selecciona la mejor solución encontrada en esta ronda, y luego se compara con la mejor solución encontrada hasta el momento; una vez que la solución de esta ronda se encuentre, si la solución es mejor, se reemplaza y registra el resultado de la solución.

8.8. Resultados

Por lo general, se establece un mecanismo de parada para que se use como condición de terminación. Dado el caso en que no se alcance la condición de parada establecida, el nuevo grupo étnico generado en el último paso se devuelve al paso de mutación y apareamiento, y luego se realizan otros pasos en secuencia. El bucle se repite continuamente hasta que se alcanza la condición de detención establecida, y finalmente se obtiene la mejor solución obtenida en todas las iteraciones. El mecanismo de detención usado es el número de iteraciones. Después de la última iteración, se genera el mejor resultado de programación encontrado en todas las iteraciones, el tiempo de finalización del resultado y el tiempo de ejecución del programa.

8.8.1. Diagrama de Gantt

La mejor solución encontrada se utiliza para dibujar el diagrama de Gantt con el objetivo de graficar la mejor solución encontrada por el algoritmo. El diagrama para el ejemplo ilustrado se presenta en la figura 8.

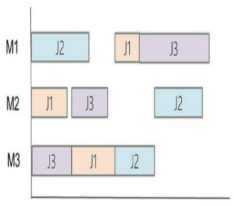

Figura 8: Diagrama de Gantt

9. Casos de prueba

Para los casos de prueba se crea un aplicativo que posibilite la planificación de la producción, integrando Excel y Python, ya que son herramientas de fácil acceso para las pequeñas y medianas empresas, las cuales no cuentan con un mecanismo a su disposición que responda a sus necesidades y proporcione una solución eficiente para sus actividades operativas, lográndose así la incorporación de métodos de solución en los ambientes tipo Job Shop. Este aplicativo funciona, inicialmente, con el ingreso de datos de entrada provenientes de un archivo de Excel; posteriormente, estos son trasladados a la base de datos de Python, y éste, a partir del algoritmo genético, busca la mejor solución posible y la selecciona para de este modo graficarla en un Diagrama de Gantt. En caso de querer utilizar esta herramienta para solucionar un problema distinto, no es necesario realizar un cambio en el código de Python; basta con cambiar los datos de entrada en el archivo de Excel. Como resultado, se optimizan los procesos productivos, tiempos de entrega y fabricación, obteniéndose a su vez mejores resultados a nivel interno de la empresa, como lo es en cuestiones económicas y administrativas.

Las técnicas tradicionales usadas por estas empresas para resolver los problemas tipo JSP no satisfacen completamente a la competencia global y el constante cambio en las exigencias de los clientes, surgiendo la necesidad de una herramienta estratégica para la programación de tareas.

Con el fin de validar la metodología propuesta e implementada, es necesario evaluar casos de prueba de la literatura especializada a partir de los cuales se pueda medir tanto la eficiencia computacional como la precisión del algoritmo para obtener los resultados. Los casos de prueba utilizados para este trabajo fueron tomados de las instancias de Taillard teniendo en cuenta problemas de tamaño 15x15 (15 trabajos y 15 máquinas). A continuación se realiza una descripción de cada una de las bases de datos tomada para la validación.

9.1. Instancia 1

Las tablas 3 y 4 representan, respectivamente, los tiempos de procesamiento de los trabajos y la secuenciación de mecanizado de la instancia 1.

Time	**O1**	**O2**	**O3**	**O4**	**O5**	**O6**	**O7**	**O8**	**O9**	**O10**	**O11**	**O12**	**O13**	**O14**	**O15**
J1	40	96	59	95	76	75	23	65	65	16	71	52	84	99	24
J2	2	88	99	52	68	13	38	35	57	37	93	38	68	94	71
J3	87	46	14	87	30	79	62	37	54	1	97	16	2	51	96
J4	19	15	42	8	72	15	76	25	78	84	62	70	81	16	97
J5	68	71	3	68	91	37	73	21	85	79	51	50	21	30	64
J6	14	1	29	72	6	31	98	50	83	2	86	33	33	98	59
J7	21	80	99	70	80	71	47	96	56	78	53	10	92	1	33
J8	29	85	89	10	30	38	38	48	16	65	90	73	88	46	47
J9	37	9	49	23	1	78	39	15	9	41	35	83	8	61	60
J10	1	73	47	46	10	37	60	84	26	11	37	79	75	49	51
J11	22	49	33	2	24	3	73	68	21	61	69	94	43	39	48
J12	81	46	21	23	86	19	64	52	22	50	11	73	77	16	75
J13	21	80	30	32	22	23	85	92	14	13	68	60	45	32	90
J14	29	95	52	59	33	12	73	96	75	12	83	3	90	57	6
J15	94	18	54	42	70	29	43	50	75	70	40	48	1	27	12

Tabla 3: Tiempo de procesamiento en instancia 1

Order	***O1***	***O2***	***O3***	***O4***	***O5***	***O6***	***O7***	***O8***	***O9***	***O10***	***O11***	***O12***	***O13***	***O14***	***O15***
J1	13	2	5	10	14	1	12	9	4	6	7	15	11	3	8
J2	6	2	15	11	14	10	7	13	9	3	8	1	4	5	12
J3	7	4	11	8	14	5	6	10	9	3	1	2	12	15	13
J4	11	8	6	1	10	14	3	9	2	15	12	4	13	7	5
J5	7	13	15	2	8	6	12	1	3	4	9	14	5	10	11
J6	5	8	7	1	9	14	13	15	4	3	6	10	11	2	12
J7	11	12	7	10	1	3	2	9	13	8	6	4	14	15	5
J8	4	11	6	7	9	5	1	15	3	8	10	12	13	2	14
J9	2	3	7	8	11	13	15	9	1	4	14	6	5	10	12
J10	13	8	7	15	4	5	1	14	11	9	12	10	6	3	2
J11	5	4	9	15	3	14	6	7	11	13	8	12	2	10	1
J12	7	13	8	6	3	5	14	12	9	1	11	4	2	10	15
J13	13	14	9	8	2	7	1	6	10	11	5	3	15	4	12
J14	13	2	5	9	7	11	8	4	1	6	14	3	10	15	12
J15	8	6	1	11	3	4	10	7	12	9	2	5	15	13	14

Tabla 4: Secuenciamiento de máquinas en instancia 1

9.2. Instancia 2

Las tablas 5 y 6 representan, respectivamente, los tiempos de procesamiento de los trabajos y la secuenciación de mecanizado de la instancia 2.

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	72	51	42	31	61	46	88	33	27	85	70	56	70	50	25
J2	19	79	79	47	40	67	43	65	84	61	30	56	19	91	68
J3	94	7	2	95	60	82	76	36	8	85	7	44	2	72	91
J4	58	67	84	34	19	19	94	41	98	96	25	40	74	88	74
J5	45	60	8	29	32	42	25	4	71	79	93	28	30	17	43
J6	84	56	46	93	66	84	40	4	15	15	54	39	77	55	31
J7	65	91	17	47	77	68	62	22	72	47	38	7	11	22	63
J8	12	21	60	42	22	84	60	52	25	53	53	56	29	83	32
J9	48	28	70	26	68	4	19	92	24	54	57	47	84	85	95
J10	36	34	65	64	30	41	53	74	44	13	41	6	32	94	37
J11	62	9	89	37	28	23	13	60	46	94	85	72	18	79	11
J12	74	61	43	26	97	62	40	60	62	78	42	8	21	11	70
J13	9	22	9	8	54	32	92	76	2	63	63	98	42	12	41
J14	67	7	91	52	87	4	1	56	82	47	35	8	92	39	11
J15	44	24	24	14	34	57	30	64	4	14	69	95	22	60	61

Tabla 5: Tiempo de procesamiento en instancia 2

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	4	8	7	15	10	9	6	5	11	2	1	13	12	3	14
J2	2	12	1	6	9	14	4	11	10	3	13	7	5	8	15
J3	8	4	9	12	1	5	10	14	2	11	7	6	15	13	3
J4	7	12	6	14	4	3	2	5	10	13	9	1	15	8	11
J5	2	12	7	6	9	8	13	10	3	15	14	4	1	5	11
J6	12	3	13	2	15	1	7	10	4	14	8	11	5	6	9
J7	15	8	12	1	13	9	10	11	4	14	5	2	3	7	6
J8	8	5	14	6	15	12	3	13	7	2	1	11	10	4	9
J9	8	5	15	11	4	1	14	9	2	3	7	13	12	10	6
J10	3	13	11	4	7	6	15	10	14	5	1	12	8	9	2
J11	12	10	4	8	7	5	1	3	6	2	11	9	14	13	15
J12	1	5	2	10	3	12	14	11	8	7	9	13	15	4	6
J13	7	1	5	11	3	15	10	14	12	13	2	9	6	4	8
J14	6	11	14	3	8	5	9	7	10	1	13	12	2	15	4
J15	12	3	1	5	15	6	10	11	7	13	9	2	8	4	14

Tabla 6: Secuenciamiento de máquinas en instancia 2

9.3. Instancia 3

Las tablas 7 y 8 representan, respectivamente, los tiempos de procesamiento de los trabajos y la secuenciación de mecanizado de la instancia 3.

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	69	81	81	62	80	3	38	62	54	66	88	82	3	12	88
J2	83	51	47	15	89	76	52	18	22	85	26	30	5	89	22
J3	62	47	93	54	38	78	71	96	19	33	44	71	90	9	21
J4	33	82	80	30	96	31	11	26	41	55	12	10	92	3	75
J5	36	49	10	43	69	72	19	65	37	57	32	11	73	89	12
J6	83	32	6	13	87	94	36	76	46	30	56	62	32	52	72
J7	29	78	21	27	17	43	14	15	16	49	72	19	99	38	64
J8	12	74	4	3	15	62	50	38	49	25	18	55	5	71	27
J9	69	13	33	47	86	31	97	48	25	40	94	22	61	59	16
J10	27	4	35	80	49	46	84	46	96	72	18	23	96	74	23
J11	36	17	81	67	47	5	51	23	82	35	96	7	54	92	38
J12	78	58	62	43	1	56	76	49	80	26	79	9	24	24	42
J13	38	86	38	38	83	36	11	17	99	14	57	64	58	96	17
J14	10	86	93	63	61	62	75	90	40	77	8	27	96	69	64
J15	73	12	14	71	3	47	84	84	53	58	95	87	90	68	75

Tabla 7: Tiempo de procesamiento en instancia 3

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	8	12	9	4	13	2	14	1	15	7	10	5	3	11	6
J2	13	2	12	10	7	4	3	5	6	9	14	15	11	1	8
J3	2	3	10	1	4	6	9	5	15	11	13	14	8	7	12
J4	14	11	7	3	15	8	5	12	1	6	10	4	9	2	13
J5	2	9	5	15	7	6	4	3	10	11	14	8	12	1	13
J6	6	15	3	13	11	2	12	5	7	10	1	14	9	4	8
J7	6	3	1	2	9	15	12	11	8	10	7	13	5	14	4
J8	5	8	11	2	10	9	3	15	12	4	6	7	14	13	1
J9	15	12	1	10	11	6	4	13	9	14	7	2	8	3	5
J10	10	1	4	11	13	14	6	2	7	15	9	12	3	8	5
J11	8	3	2	13	4	15	5	7	6	10	9	14	11	1	12
J12	1	9	15	13	10	6	7	11	8	12	4	5	2	14	3
J13	9	13	11	12	15	4	7	2	5	6	1	10	14	3	8
J14	14	3	12	1	15	11	4	2	13	5	6	7	8	10	9
J15	2	14	1	12	3	11	5	9	4	6	8	7	10	13	15

Tabla 8: Secuenciamiento de máquinas en instancia 3

9.4. Instancia 4

Las tablas 9 y 10 representan, respectivamente, los tiempos de procesamiento de los trabajos y la secuenciación de mecanizado de la instancia 4.

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	86	60	10	59	65	94	71	25	98	49	43	8	90	21	73
J2	68	28	38	36	93	35	37	28	62	86	65	11	20	82	23
J3	33	67	96	91	83	81	60	88	20	62	22	79	38	40	82
J4	13	14	73	88	24	16	78	70	53	68	73	90	58	7	4
J5	93	52	63	13	19	41	71	59	19	60	85	99	73	95	19
J6	62	60	93	16	10	72	88	69	58	41	46	63	76	83	62
J7	50	68	90	34	44	5	8	25	70	53	78	92	62	85	70
J8	60	64	92	44	63	91	21	1	96	19	59	12	41	11	94
J9	93	46	51	37	91	90	63	40	68	13	16	83	49	24	23
J10	5	35	21	14	66	3	6	98	63	64	76	94	17	62	37
J11	35	42	62	68	73	27	52	39	41	25	9	34	50	41	98
J12	23	32	35	10	29	68	20	8	58	62	39	32	8	33	91
J13	28	31	3	28	66	59	24	45	81	8	44	42	2	23	53
J14	11	93	27	59	62	23	23	7	77	64	60	97	36	53	72
J15	36	98	38	24	84	47	72	1	91	85	68	42	20	30	30

Tabla 9: Tiempo de procesamiento en instancia 4

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	10	15	5	14	11	4	8	9	1	6	2	3	13	7	12
J2	11	9	12	15	4	14	10	8	5	3	7	2	6	13	1
J3	8	1	7	6	15	14	3	12	5	13	2	10	4	11	9
J4	10	12	15	1	2	9	6	11	13	5	14	4	7	8	3
J5	12	5	14	4	9	2	11	13	3	15	7	8	1	10	6
J6	6	3	2	11	1	5	9	15	7	4	10	8	12	13	14
J7	6	11	14	1	10	9	2	12	15	8	13	3	7	5	4
J8	13	1	10	4	14	7	6	8	3	15	12	9	11	2	5
J9	12	11	6	14	2	10	9	8	4	7	1	3	15	13	5
J10	3	15	4	11	7	2	1	14	12	5	6	9	8	13	10
J11	12	15	14	6	5	10	2	7	13	1	3	9	11	4	8
J12	13	4	11	9	5	8	14	12	15	2	3	1	6	7	10
J13	9	14	6	1	12	10	5	13	2	11	7	3	8	15	4
J14	3	6	5	4	10	2	12	14	8	7	11	15	1	9	13
J15	2	11	5	3	1	8	7	10	12	13	6	15	4	14	9

Tabla 10: Secuenciamiento de máquinas en instancia 4

9.5. Instancia 5

Las tablas 11 y 12 representan, respectivamente, los tiempos de procesamiento de los trabajos y la secuenciación de mecanizado de la instancia 5.

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	94	66	10	53	26	15	65	82	10	27	93	92	96	70	83
J2	74	31	88	51	57	78	8	7	91	79	18	51	18	99	33
J3	4	82	40	86	50	54	21	6	54	68	82	20	39	35	68
J4	73	23	30	30	53	94	58	93	32	91	30	56	27	92	9
J5	78	23	21	60	36	29	95	99	79	76	93	42	52	42	96
J6	29	61	88	70	16	31	65	83	78	26	50	87	62	14	30
J7	18	75	20	4	91	68	19	54	85	73	43	24	37	87	66
J8	32	52	9	49	61	35	99	62	6	62	7	80	3	57	7
J9	85	30	96	91	13	87	82	83	78	56	85	8	66	88	15
J10	5	59	30	60	41	17	66	89	78	88	69	45	82	6	13
J11	90	27	1	8	91	80	89	49	32	28	90	93	6	35	73
J12	47	43	75	8	51	3	84	34	28	60	69	45	67	58	87
J13	65	62	97	20	31	33	33	77	50	80	48	90	75	96	44
J14	28	21	51	75	17	89	59	56	63	18	17	30	16	7	35
J15	57	16	42	34	37	26	68	73	5	8	12	87	83	20	97

Tabla 11: Tiempo de procesamiento en instancia 5

Time	*O1*	*O2*	*O3*	*O4*	*O5*	*O6*	*O7*	*O8*	*O9*	*O10*	*O11*	*O12*	*O13*	*O14*	*O15*
J1	7	13	5	8	4	3	11	12	9	15	10	14	6	1	2
J2	5	6	8	15	14	9	12	10	7	11	1	4	13	2	3
J3	2	9	10	13	7	12	14	6	1	3	8	11	5	4	15
J4	6	3	10	7	11	1	14	5	8	15	12	9	13	2	4
J5	8	9	7	11	5	10	3	15	13	6	2	14	12	1	4
J6	6	4	13	14	12	5	15	8	3	2	11	1	10	7	9
J7	13	4	8	9	15	7	2	12	5	6	3	11	1	14	10
J8	12	6	1	8	13	14	15	2	3	9	5	4	10	7	11
J9	11	12	7	15	1	2	3	6	13	5	9	8	10	14	4
J10	7	12	10	3	9	1	14	4	11	8	2	13	15	5	6
J11	5	8	14	1	6	13	7	9	15	11	4	2	12	10	3
J12	3	15	1	13	7	11	8	6	9	10	14	2	4	12	5
J13	6	9	11	3	4	7	10	1	14	5	2	12	13	8	15
J14	9	15	5	14	6	7	10	2	13	8	12	11	4	3	1
J15	11	9	13	7	5	2	14	15	12	1	8	4	3	10	6

Tabla 12: Secuenciamiento de máquinas en instancia 5

10. Resultados

La metodología se validó inicialmente con una población generada totalmente aleatoria. En cada uno de los casos la semilla permaneció intacta en cada instancia estudiada. Los resultados obtenidos en las corridas de prueba se presentan en la Tabla 13.

Tamaño de la población: 5
Tasa de cruce: 0.2
Tasa de mutación:0.2
Tasa de selección:0.2
Número de iteraciones: 5000

Instancias	Secuencia optima	Función Objetivo	Mejor solución encontrada	Tiempo(seg)
Instancia 1	[12, 1, 1, 0, 5, 14, 1, 11, 4, 5, 10, 14, 7, 9, 2, 12, 8, 14, 3, 14, 11, 8, 1, 0, 12, 0, 2, 14, 2, 6, 6, 8, 14, 1, 7, 2, 10, 10, 7, 6, 9, 1, 12, 9, 12, 8, 5, 2, 5, 7, 5, 11, 6, 1, 10, 12, 6, 9, 7, 6, 13, 3, 12, 2, 10, 2, 6, 4, 13, 11, 4, 12, 5, 5, 3, 0, 8, 3, 13, 6, 8, 11, 14, 12, 9, 10, 11, 5, 6, 11, 0, 14, 0, 10, 3, 1, 11, 9, 7, 4, 9, 8, 12, 11, 10, 3, 13, 0, 8, 0, 7, 13, 2, 0, 5, 8, 11, 3, 1, 4, 6, 9, 14, 13, 1, 6, 5, 4, 10, 7, 8, 13, 10, 11, 3, 5, 9, 7, 7, 0, 2, 7, 10, 4, 3, 7, 4, 8, 2, 11, 6, 1, 7, 14, 8, 12, 2, 3, 8, 11, 9, 6, 13, 8, 14, 14, 10, 2, 14, 12, 13, 12, 4, 9, 9, 5, 0, 7, 1, 10, 14, 11, 1, 6, 3, 2, 4, 4, 2, 0, 4, 4, 4, 9, 5, 0, 7, 3, 1, 5, 13, 13, 2, 0, 9, 3, 12, 8, 14, 10, 10, 3, 12, 4, 6, 1, 0, 9, 13, 13, 11, 13, 5, 3, 13]	1846	1224	88.813
Instancia 2	[11, 11, 5, 3, 1, 3, 12, 1, 5, 1, 7, 14, 6, 0, 1, 14, 7, 5, 3, 7, 13, 13, 5, 14, 9, 0, 14, 10, 1, 6, 13, 4, 12, 3, 0, 12, 13, 2, 13, 2, 1, 4, 5, 8, 1, 6, 2, 8, 8, 5, 12, 14, 9, 7, 1, 0, 5, 3, 10, 4, 9, 13, 2, 4, 11, 11, 6, 3, 2, 7, 9, 0, 3, 14, 6, 7, 10, 11, 5, 11, 8, 4, 14, 10, 10, 14, 0, 4, 11, 0, 7, 5, 7, 12, 2, 11, 8, 3, 7, 6, 10, 0, 1, 6, 13, 0, 7, 12, 6, 10, 2, 3, 14, 4, 6, 9, 12, 9, 10, 12, 5, 3, 9, 2, 10, 8, 11, 9, 13, 11, 7, 4, 13, 7, 3, 9, 2, 5, 4, 2, 2, 7, 5, 9, 2, 14, 3, 7, 0, 4, 4, 10, 8, 9, 0, 6, 8, 10, 7, 13, 0, 10, 11, 2, 6, 9, 8, 1, 13, 10, 12, 12, 6, 4, 4, 13, 2, 10, 1, 6, 2, 11, 14, 1, 9, 13, 6, 4, 0, 14, 8, 12, 14, 5, 8, 1, 13, 11, 8, 3, 5, 0, 11, 12, 14, 8, 1, 8, 12, 3, 0, 10, 11, 6, 9, 3, 9, 1, 4, 14, 13, 12, 12, 8, 5]	1812	1175	83.994
Instancia 3	[2, 6, 5, 4, 2, 13, 7, 9, 2, 3, 8, 13, 1, 8, 6, 1, 12, 10, 1, 0, 9, 8, 1, 9, 4, 4, 8, 1, 11, 9, 2, 12, 0, 5, 13, 3, 11, 14, 9, 6, 10, 12, 5, 2, 11, 7, 12, 1, 8, 12, 9, 4, 5, 9, 0, 7, 7, 7, 2, 10, 8, 6, 11, 6, 1, 6, 10, 3, 0, 10, 14, 12, 8, 14, 9, 11, 2, 8, 7, 12, 5, 2, 4, 9, 10, 8, 7, 10, 14, 6, 9, 5, 10, 14, 13, 11, 0, 5, 3, 14, 12, 5, 2, 10, 4, 4, 2, 8, 8, 13, 2, 10, 5, 6, 10, 6, 13, 4, 7, 5, 0, 6, 0, 8, 3, 10, 1, 4, 8, 11, 7, 12, 14, 9, 5, 1, 1, 3, 10, 11, 3, 2, 12, 12, 14, 13, 13, 5, 9, 4, 10, 3, 10, 11, 6, 4, 13, 14, 9, 4, 4, 6, 1, 0, 8, 12, 13, 7, 2, 7, 12, 6, 7, 3, 14, 4, 14, 11, 13, 1, 11, 0, 6, 1, 3, 1, 9, 5, 0, 11, 4, 0, 0, 0, 11, 7, 0, 2, 0, 14, 8, 3, 14, 13, 12, 14, 1, 14, 13, 13, 3, 12, 2, 7, 3, 11, 3, 9, 5, 13, 11, 3, 7, 6, 5]	1829	1218	96.946
Instancia 4	[14, 9, 8, 6, 12, 4, 14, 2, 14, 11, 10, 9, 14, 0, 14, 5, 12, 4, 11, 14, 6, 8, 8, 7, 10, 11, 3, 12, 13, 14, 10, 11, 6, 12, 13, 12, 5, 13, 1, 14, 9, 6, 0, 6, 0, 7, 0, 1, 5, 3, 12, 11, 9, 7, 7, 3, 0, 14, 13, 1, 11, 3, 5, 9, 2, 6, 1, 5, 0, 0, 5, 9, 2, 10, 1, 1, 9, 3, 10, 2, 10, 0, 12, 6, 11, 3, 6, 4, 14, 10, 13, 3, 4, 0, 14, 3, 7, 3, 8, 8, 4, 2, 2, 5, 12, 3, 4, 9, 5, 0, 13, 10, 1, 14, 13, 6, 6, 11, 10, 0, 11, 9, 10, 14, 1, 7, 2, 5, 5, 10, 3, 4, 9, 8, 6, 11, 3, 13, 6, 3, 12, 1, 11, 7, 1, 0, 9, 10, 2, 8, 1, 10, 13, 1, 1, 13, 7, 2, 4, 7, 5, 7, 4, 9, 6, 2, 8, 8, 9, 5, 7, 13, 4, 12, 11, 0, 8, 11, 2, 11, 8, 0, 13, 9, 14, 3, 2, 4, 7, 6, 12, 10, 6, 13, 12, 3, 8, 5, 1, 7, 5, 2, 5, 2, 9, 2, 4, 13, 12, 1, 11, 8, 7, 13, 12, 14, 7, 12, 10, 4, 0, 8, 4, 4, 8]	1808	1244	87.674
Instancia 5	[7, 11, 5, 3, 6, 5, 9, 12, 14, 12, 0, 0, 10, 13, 6, 10, 12, 10, 13, 11, 9, 11, 6, 13, 9, 13, 6, 0, 12, 6, 10, 14, 12, 3, 4, 4, 3, 4, 4, 14, 1, 10, 11, 7, 7, 3, 14, 13, 7, 7, 12, 11, 14, 5, 0, 11, 14, 8, 8, 14, 14, 10, 5, 9, 11, 13, 12, 14, 6, 5, 8, 9, 5, 9, 0, 4, 9, 12, 4, 7, 0, 13, 7, 1, 5, 3, 13, 3, 2, 6, 1, 5, 11, 1, 3, 8, 0, 9, 5, 1, 2, 7, 7, 7, 12, 1, 4, 14, 8, 0, 7, 4, 5, 0, 2, 12, 1, 10, 8, 1, 3, 2, 10, 13, 8, 6, 11, 0, 3, 7, 8, 4, 9, 3, 3, 4, 13, 0, 12, 6, 14, 9, 1, 9, 7, 1, 8, 8, 10, 2, 2, 14, 2, 6, 8, 2, 14, 11, 4, 0, 12, 10, 3, 5, 2, 13, 10, 12, 9, 6, 0, 8, 4, 14, 14, 13, 10, 2, 12, 4, 0, 11, 8, 2, 12, 11, 7, 1, 6, 2, 6, 7, 13, 9, 8, 9, 1, 4, 10, 5, 1, 2, 6, 3, 0, 13, 5, 8, 3, 2, 4, 2, 5, 10, 3, 10, 5, 11, 11, 6, 13, 1, 11, 1, 4, 9]	1808	1231	99.946

Tabla 13: Resultados generales en cada instancia

10.1. Resultado en instancia 1

La figura 9 representa un comparativo entre el makespan obtenido por el algoritmo genético durante cada generación y la figura 10 el diagrama de Gantt con la mejor solución encontrada para la instancia 1.

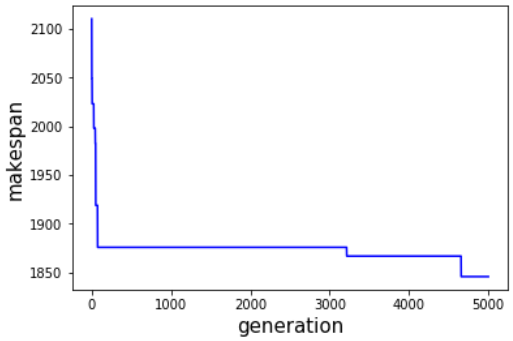

Figura 9: Makespan VS Generaciones en instancia 1

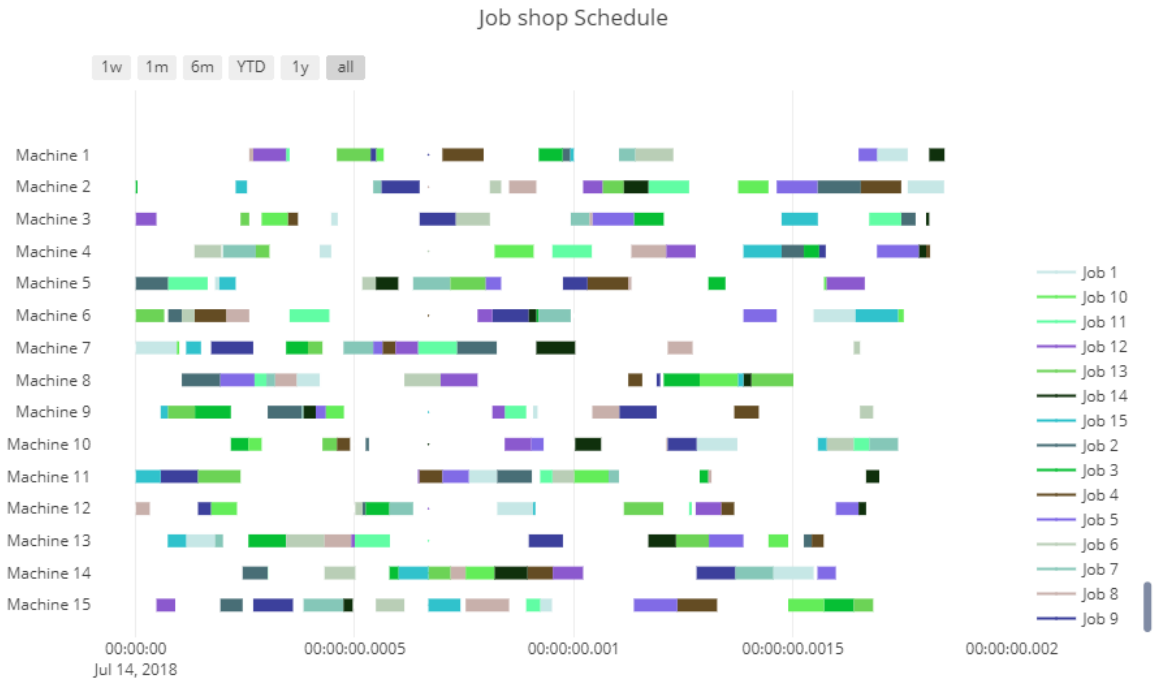

Figura 10: Diagrama de Gantt para la instancia 1

10.2. Resultado en instancia 2

La figura 11 representa un comparativo entre el makespan obtenido por el algoritmo genético durante cada generación y la figura 12 el diagrama de Gantt con la mejor solución encontrada para la instancia 2.

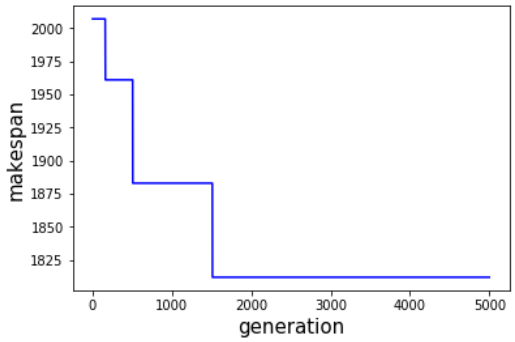

Figura 11: Makespan VS Generaciones en instancia 2

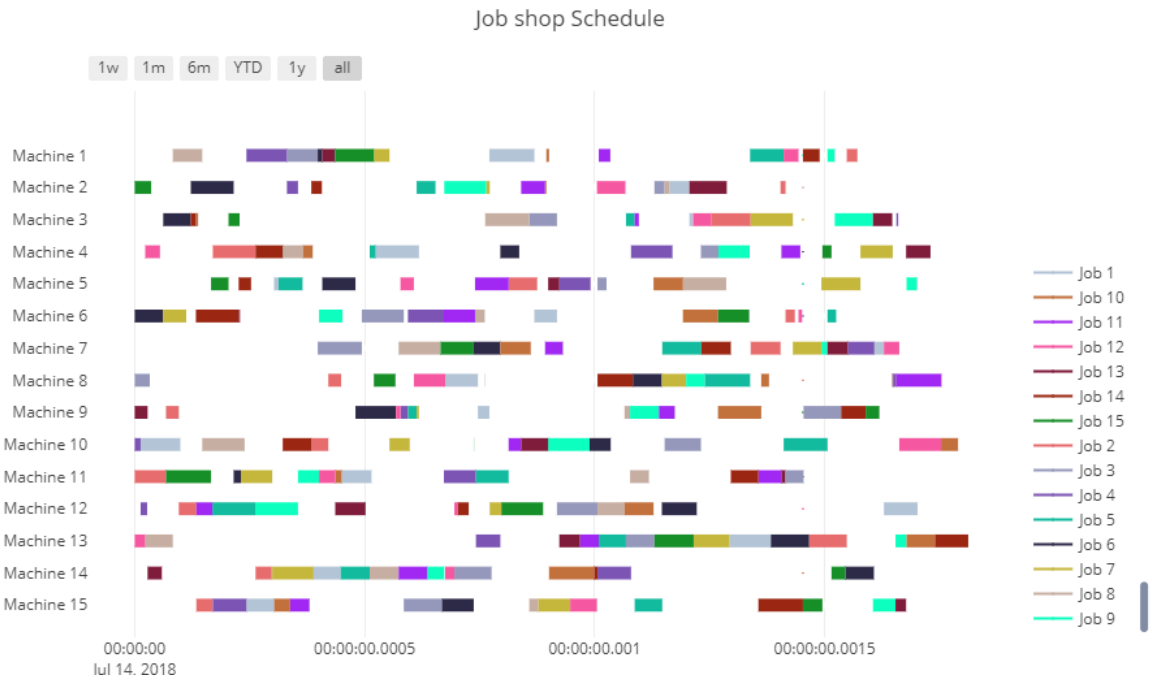

Figura 12: Diagrama de Gantt para la instancia 2

10.3. Resultado en instancia 3

La figura 13 representa un comparativo entre el makespan obtenido por el algoritmo genético durante cada generación y la figura 14 el diagrama de Gantt con la mejor solución encontrada para la instancia 3.

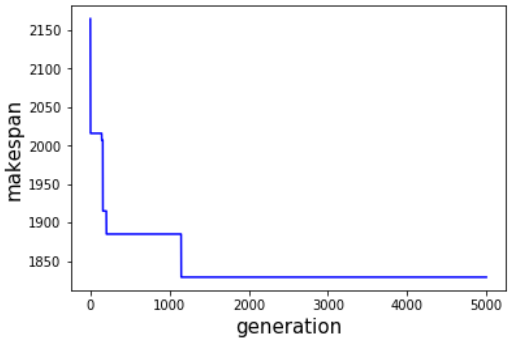

Figura 13: Makespan VS Generaciones en instancia 3

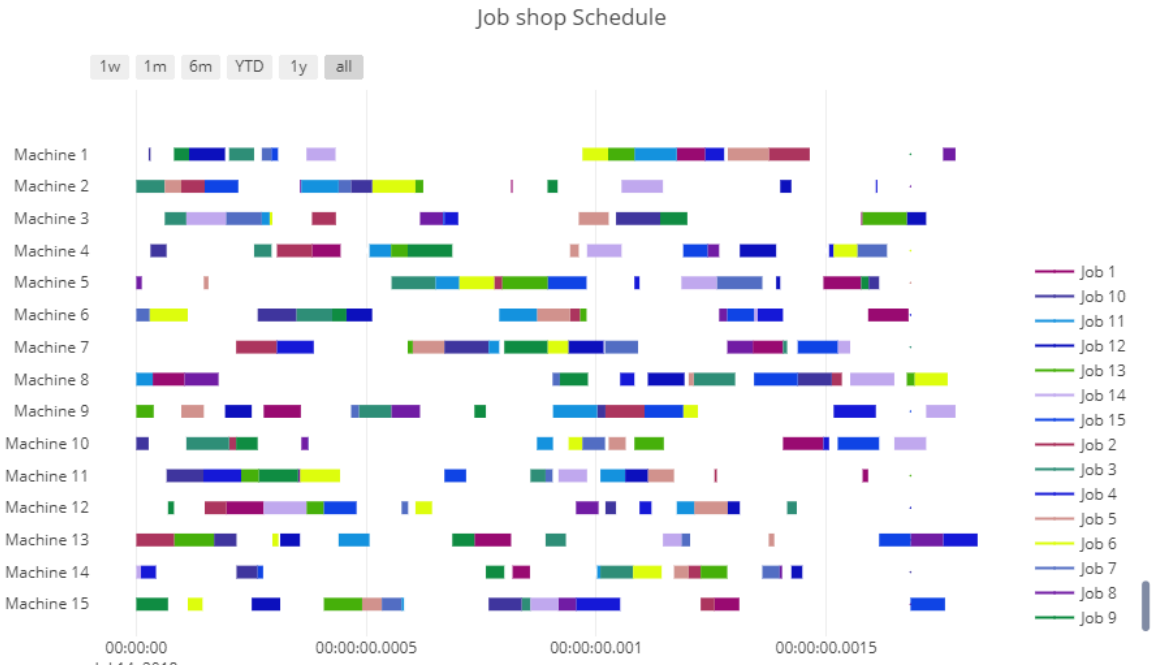

Figura 14: Diagrama de Gantt para la instancia 3

10.4. Resultado en instancia 4

La figura 15 representa un comparativo entre el makespan obtenido por el algoritmo genético durante cada generación y la figura 16 el diagrama de Gantt con la mejor solución encontrada para la instancia 4.

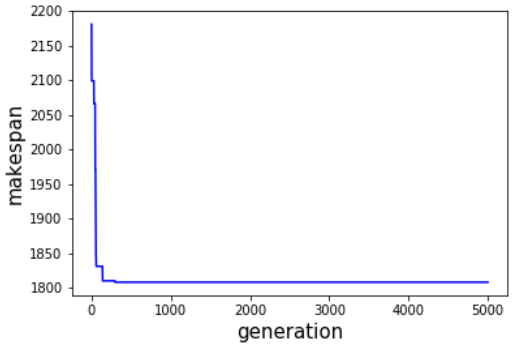

Figura 15: Makespan VS Generaciones en instancia 4

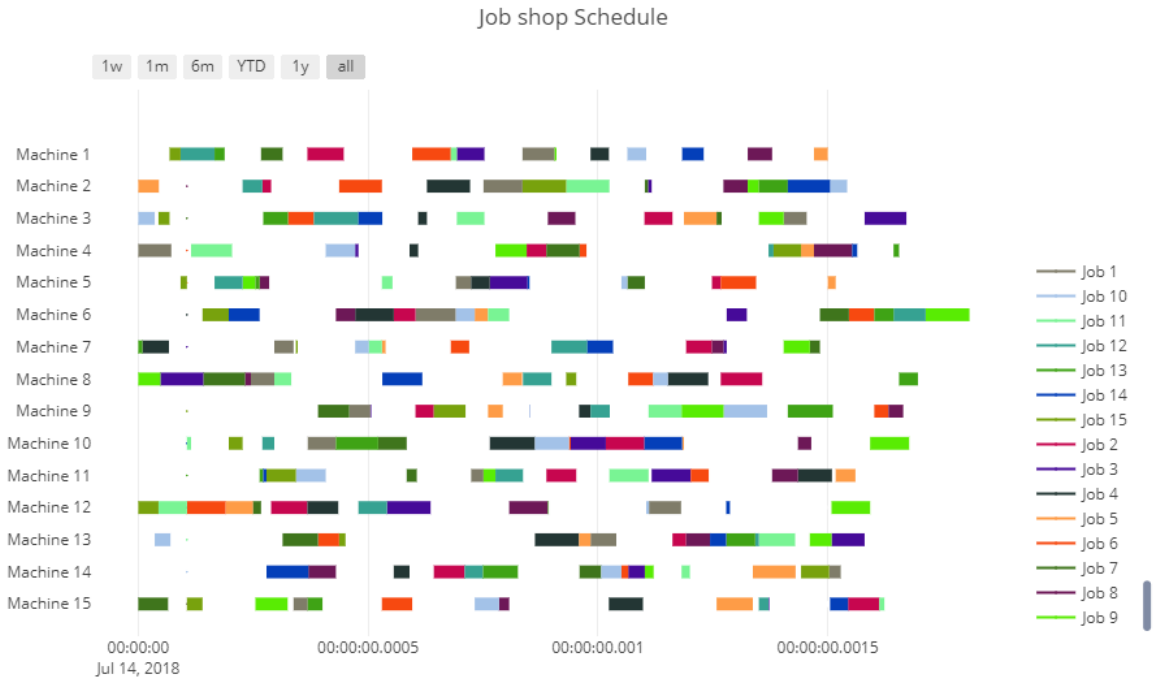

Figura 16: Diagrama de Gantt para la instancia 4

10.5. Resultado en instancia 5

La figura 17 representa un comparativo entre el makespan obtenido por el algoritmo genético durante cada generación y la figura 18 el diagrama de Gantt con la mejor solución encontrada para la instancia 4.

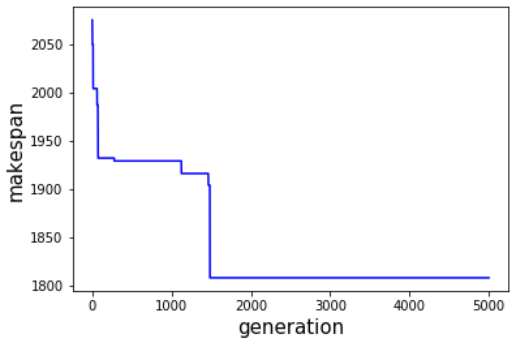

Figura 17: Makespan VS Generaciones en instancia 5

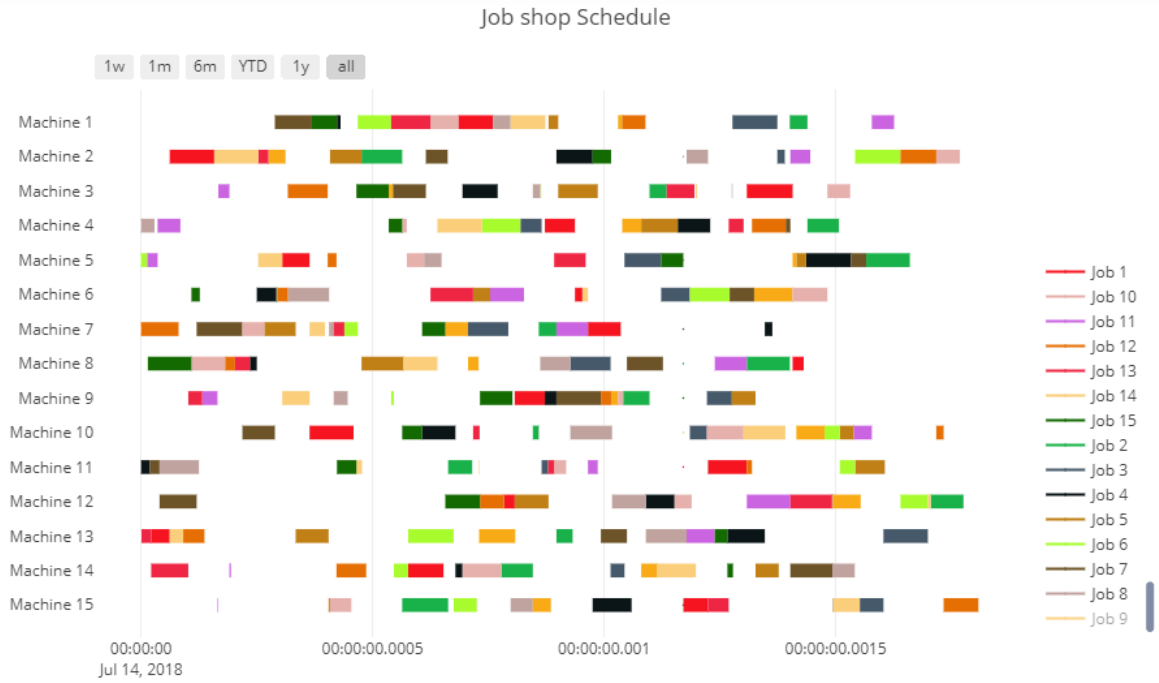

Figura 18: Diagrama de Gantt para la instancia 5

11. Conclusiones

- Se desarrolló una metodología basada en la técnica metaheurística conocida como el algoritmo genético para dar solución al problema Job Shop (JSP), teniendo en cuenta como medida de efectividad la optimización del tiempo total requerido para terminar todas las tareas (Makespan).

- Esta investigación hace un aporte tanto en el ámbito práctico, ya que brindará a las pequeñas y medianas empresas una herramienta accesible para encontrar una solución eficiente a este tipo de problemas de planificación, como en el ámbito académico, pues servirá como punto de partida para trabajos futuros donde se presenten modificaciones a la propuesta actual o la comparación de la eficiencia computacional de diferentes técnicas para solucionar el mismo problema.

- Se decidió usar una codificación de tipo indirecto debido a las características del JSP, ya que se trata de uno de los problemas combinacionales más difíciles, las experiencias expuestas en la literatura especializada y las ventajas se adecuaban más al tipo de problema que se deseaba resolver. Emplear esta codificación para la representación en el algoritmo genético facilita la implementación de los operadores genéticos además del control de la factibilidad del problema.

- Utilizar la integración del lenguaje de programación Python y Microsoft Excel permitió crear una herramienta de fácil acceso para las pequeñas y medianas empresas, la cual brinda una solución eficiente al problema junto con una interfaz gráfica de la misma. Por otro lado, dicha integración significa más complejidad computacional y represeta más tiempo de procesamiento en el algoritmo y, por tanto, en la solución del JSP.

Referencias

[1] R. A. G. Rendon, A. H. E. Zuluaga, and E. M. T. Ocampo, *Técnicas metaheuristicas de optimización*. Universidad Tecnologica de Pereira, 2008.

[2] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[3] M. S. Garza Solís, "Aplicación de modelos de producción para el programa del curso sistemas de producción a nivel licenciatura," Ph.D. dissertation, Universidad Autónoma de Nuevo León, 2000.

[4] D. Sipper and R. L. Bulfin, "Planeación y control de la producción," McGraw-Hill, Tech. Rep., 1999.

[5] M. Abdolrazzagh-Nezhad and S. Abdullah, "Job shop scheduling: Classification, constraints and objective functions," *International Journal of Computer and Information Engineering*, vol. 11, no. 4, pp. 429–434, 2017.

[6] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval research logistics quarterly*, vol. 1, no. 1, pp. 61–68, 1954.

[7] J. R. Barker and G. B. McMahon, "Scheduling the general job-shop," *Management Science*, vol. 31, no. 5, pp. 594–598, 1985.

[8] P. Brucker and P. Brucker, *Scheduling algorithms*. Springer, 2007, vol. 3.

[9] S. B. Akers Jr, "Letter to the editor—a graphical approach to production scheduling problems," *Operations Research*, vol. 4, no. 2, pp. 244–245, 1956.

[10] N. Hefetz and I. Adiri, "An efficient optimal algorithm for the two-machines unit-time jobshop schedule-length problem," *Mathematics of Operations Research*, vol. 7, no. 3, pp. 354–360, 1982.

[11] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Mathematics of operations research*, vol. 1, no. 2, pp. 117–129, 1976.

[12] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[13] H. Fisher, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, pp. 225–251, 1963.

[14] J. Carlier and É. Pinson, "An algorithm for solving the job-shop problem," *Management science*, vol. 35, no. 2, pp. 164–176, 1989.

[15] S. French, "Sequencing and scheduling," *An Introduction to the Mathematics of the Job-shop*, 1982.

[16] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[17] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Operations research*, vol. 8, no. 4, pp. 487–503, 1960.

[18] G. L. Nemhauser and L. A. Wolsey, "Integer and combinatorial optimization john wiley & sons," *New York*, vol. 118, 1988.

[19] J. Blazewicz, M. Dror, and J. Weglarz, "Mathematical programming formulations for machine scheduling: A survey," *European Journal of Operational Research*, vol. 51, no. 3, pp. 283–300, 1991.

[20] B. Lageweg, J. Lenstra, and A. Rinnooy Kan, "Job-shop scheduling by implicit enumeration," *Management Science*, vol. 24, no. 4, pp. 441–450, 1977.

[21] P. Brucker, "An efficient algorithm for the job-shop problem with two jobs," *Computing*, vol. 40, no. 4, pp. 353–359, 1988.

[22] P. Brucker and B. Jurisch, "A new lower bound for the job-shop scheduling problem," *European Journal of Operational Research*, vol. 64, no. 2, pp. 156–167, 1993.

[23] C. N. Potts, "Analysis of a heuristic for one machine sequencing with release dates and delivery times," *Operations Research*, vol. 28, no. 6, pp. 1436–1441, 1980.

[24] G. H. Brooks, "An algorithm for finding optimal or near optimal solutions to the production scheduling problem," *The Journal of Industrial Engineering*, vol. 16, no. 1, pp. 34–40, 1969.

[25] E. Ignall and L. Schrage, "Application of the branch and bound technique to some flow-shop scheduling problems," *Operations research*, vol. 13, no. 3, pp. 400–412, 1965.

[26] Z. Lomnicki, "A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem," *Journal of the operational research society*, vol. 16, no. 1, pp. 89–100, 1965.

[27] E. Balas, "Machine sequencing via disjunctive graphs: an implicit enumeration algorithm," *Operations research*, vol. 17, no. 6, pp. 941–957, 1969.

[28] J. Carlier, "The one-machine sequencing problem," *European Journal of Operational Research*, vol. 11, no. 1, pp. 42–47, 1982.

[29] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete applied mathematics*, vol. 49, no. 1-3, pp. 107–127, 1994.

[30] J. Grabowski, E. Nowicki, and S. Zdrzałka, "A block approach for single-machine scheduling with release dates and due dates," *European Journal of Operational Research*, vol. 26, no. 2, pp. 278–285, 1986.

[31] M. Perregaard and J. Clausen, "Parallel branch-and-bound methods for thejob-shop scheduling problem," *Annals of Operations Research*, vol. 83, pp. 137–160, 1998.

[32] J. Carlier and E. Pinson, "Adjustment of heads and tails for the job-shop problem," *European Journal of Operational Research*, vol. 78, no. 2, pp. 146–161, 1994.

[33] P. D. Martin, "A time-oriented approach to computing optimal schedules for the job-shop scheduling problem." 1997.

[34] F. Glover and H. J. Greenberg, "New approaches for heuristic search: A bilateral linkage with artificial intelligence," *European Journal of Operational Research*, vol. 39, no. 2, pp. 119–130, 1989.

[35] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Operations research*, vol. 25, no. 1, pp. 45–61, 1977.

[36] R. Haupt, "A survey of priority rule-based scheduling," *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3–16, 1989.

[37] F. Viviers, "A decision support system for job shop scheduling," *European Journal of Operational Research*, vol. 14, no. 1, pp. 95–103, 1983.

[38] W. B. Crowston, F. Glover, J. D. Trawick *et al.*, "Probabilistic and parametric learning combinations of local job shop scheduling rules," CARNEGIE INST OF TECH PITTSBURGH PA GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION, Tech. Rep., 1963.

[39] S. Lawrence, "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[40] T. Morton and D. W. Pentico, *Heuristic scheduling systems: with applications to production systems and project management.* John Wiley & Sons, 1993, vol. 3.

[41] M. L. Fisher and A. H. Rinnooy Kan, "The design, analysis and implementation of heuristics," *Management Science*, vol. 34, no. 3, pp. 263–265, 1988.

[42] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management science*, vol. 34, no. 3, pp. 391–401, 1988.

[43] K. Fukumori, "Fundamental scheme for train scheduling (application of range-constriction search)." MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB, Tech. Rep., 1980.

[44] M. Fox, "Constraint-directed search: A case study of job-shop scheduling, mor," 1987.

[45] M. S. Fox and N. M. Sadeh, "Why is scheduling difficult? a csp perspective." in *ECAI*, 1990, pp. 754–767.

[46] W. P. Nuijten and E. H. Aarts, "A computational study of constraint satisfaction for multiple capacitated job shop scheduling," *European Journal of Operational Research*, vol. 90, no. 2, pp. 269–284, 1996.

[47] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[48] W. D. Harvey and M. L. Ginsberg, "Limited discrepancy search," in *IJCAI (1)*, 1995, pp. 607–615.

[49] H.-C. Zhang and S. Huang, "Applications of neural networks in manufacturing: a state-of-the-art survey," *The International Journal of Production Research*, vol. 33, no. 3, pp. 705–728, 1995.

[50] J. J. Hopfield and D. W. Tank, ""neural" computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[51] F. Y.-P. Simon *et al.*, "Stochastic neural networks for solving job-shop scheduling. i. problem representation," in *IEEE 1988 International Conference on Neural Networks*. IEEE, 1988, pp. 275–282.

[52] M. M. Van Hulle, "A goal programming network for mixed integer linear programming: a case study for the job-shop scheduling problem," *International Journal of Neural Systems*, vol. 2, no. 03, pp. 201–209, 1991.

[53] C.-C. Lo and C.-C. Hsu, "A parallel distributed processing technique for job-shop scheduling problems," in *Proceedings of 1993 International Conference on Neural Networks (IJCNN-93-Nagoya, Japan)*, vol. 2. IEEE, 1993, pp. 1602–1605.

[54] M. Dorigo, M. Dorigo, V. Manjezzo, and M. Trubian, "Ant system for job-shop scheduling," *Belgian Journal of Operations Research*, vol. 34, pp. 39–53, 1994.

[55] J.-L. Deneubourg, J. M. Pasteels, and J.-C. Verhaeghe, "Probabilistic behaviour in ants: a strategy of errors?" *Journal of theoretical Biology*, vol. 105, no. 2, pp. 259–271, 1983.

[56] U. Dorndorf and E. Pesch, "Evolution based learning in a job shop scheduling environment," *Computers & Operations Research*, vol. 22, no. 1, pp. 25–40, 1995.

[57] J. R. Evans, "Structural analysis of local search heuristics in combinatorial optimization," *Computers & operations research*, vol. 14, no. 6, pp. 465–477, 1987.

[58] H. Matsuo, "A controlled search simulated annealing method for the general jobshop scheduling problem," *Technical Report Working Paper 03-04-88*, 1988.

[59] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[60] R. S. Barr, B. L. Golden, J. P. Kelly, M. G. Resende, and W. R. Stewart, "Designing and reporting on computational experiments with heuristic methods," *Journal of heuristics*, vol. 1, no. 1, pp. 9–32, 1995.

[61] F. Glover, "Tabu search and adaptive memory programming—advances, applications and challenges," in *Interfaces in computer science and operations research*. Springer, 1997, pp. 1–75.

[62] H. R. Lourenço and M. Zwijnenburg, "Combining the large-step optimization with

tabu-search: Application to the job-shop scheduling problem," in *Meta-Heuristics*. Springer, 1996, pp. 219–236.

[63] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[64] V. Černỳ, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of optimization theory and applications*, vol. 45, no. 1, pp. 41–51, 1985.

[65] H. Holland John, "Adaptation in natural and artificial systems," *Ann Arbor: University of Michigan Press*, 1975.

[66] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[67] E. Falkenauer and S. Bouffouix, "A genetic algorithm for job shop," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation.* IEEE, 1991, pp. 824–829.

[68] F. Della Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 15–24, 1995.

[69] S. Kobayashi, "An efficient genetic algorithm for job-shop scheduling problems," in *Proc. of 6th Int. Conf. on Genetic Algorithms, 1995*, 1995.

[70] H. Tamaki, "A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling," *Parallel Problem Solving from Nature 2*, pp. 573–582, 1992.

[71] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems." in *PPSN*, vol. 2, 1992, pp. 281–290.

[72] C. Bierwirth, "A generalized permutation approach to job shop scheduling with genetic algorithms," *Operations-Research-Spektrum*, vol. 17, no. 2-3, pp. 87–92, 1995.

[73] D. C. Mattfeld, C. Bierwirth, and H. Kopfer, "A search space analysis of the job shop scheduling problem," *Annals of Operations Research*, vol. 86, pp. 441–453, 1999.

[74] T. Yamada and R. Nakano, "Job-shop scheduling by simulated annealing combined with deterministic local search," in *Meta-Heuristics*. Springer, 1996, pp. 237–248.

[75] M. Gen, Y. Tsujimura, and E. Kubota, "Solving job-shop scheduling problems by genetic algorithm," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 2. IEEE, 1994, pp. 1577–1582.

[76] R. Gallego, A. Escobar, E. Toro, and R. Romero, "Técnicas heurísticas y metaheurísticas de optimización," *Editorial Universidad Tecnológica de Pereira, ISBN: 9789587222074, Pereira*, 2015.